Natural Language Processing and Applications

Unit 1

Text Pre-processing and Language Modelling

Knowledge in language processing - NLP applications - regular expressions - words - corpora - text normalization - minimum edit distance - n-gram language models - neural language models - RNNs as language models

* Natural Language Processing

→ NLP is a subset of artificial intelligence, computer science and linguistics focused on making human communication, such as speech and text comprehensible to computers

Reasons - develop automated tools for language processing
- gain a better understanding of human communication

Requires - knowledge of how humans acquire, store and process language
- knowledge of the world and language

* Origins of NLP

- Natural Language Understanding - involves only interpretation of language
- Natural Language Processing - involves both understanding (interpretation)

and generation (production)

- includes both speech and text processing

- **Computational Linguistics**

- concerned with the study of language using computational models of linguistic phenomena

- deals with knowledge representations

- is of two categories:

A. **Knowledge-driven** - expressed as a set of handcrafted grammar rules

→ The disadvantage is a knowledge bottleneck.

B. **Data-driven** - presume the existence of a large amount of data and employ some machine learning technique to learn patterns.

→ The disadvantage is that it is dependent on the quantity of data.

* **Applications of NLP**

mostly solved	making good progress	still really hard
(i) spam detection	(i) sentiment analysis	(i) question answering
(ii) POS tagging	(ii) coreference resolution	(ii) paraphrasing
(iii) named entity recognition	(iii) word sense disambiguation (mouse vs. mouse)	(iii) summarizing
	(iv) parsing	(iv) dialog
	(v) machine translation	
	(vi) information extraction (IE)	

A. **Machine Translation**

- translates from one human language to another
- requires understanding of phrases, grammar, semantics of the language.

B. **Speech Recognition**

- mapping speech signals to a set of words
- requires knowledge of homonyms and acoustic ambiguities (like deer, dear)

C. **Speech Synthesis**

- automatic production of speech

D. **Natural language interfaces to database.**

- querying structural database using natural language sentences

E. **Information Retrieval**

- identify documents that are relevant to a user's query
- requires indexing, word sense disambiguation, query modification/expansion

F. **Information Extraction**

- captures and outputs factual information, input is based upon pre-defined schemas and templates

G. **Question Answering**

- attempts to find a precise answer unlike information retrieval

11. Text Summarization

- creation of summaries of documents
- involves syntactic, semantic and discourse level processing

* Early NLP Systems

(i) ELIZA - program imitating the responses of a psycho therapist
- used simple pattern matching, without any deeper knowledge of the world or conversation

(ii) SysTran (System Translation) - russian-english translator

(iii) Taum Meteo - generates whether reports are in English or French

(iv) SHRDLU - NL understanding system that simulates the actions of a robot in a block world domain
- uses syntactic parsing and semantic reasoning to understand instructions for manipulating the blocks

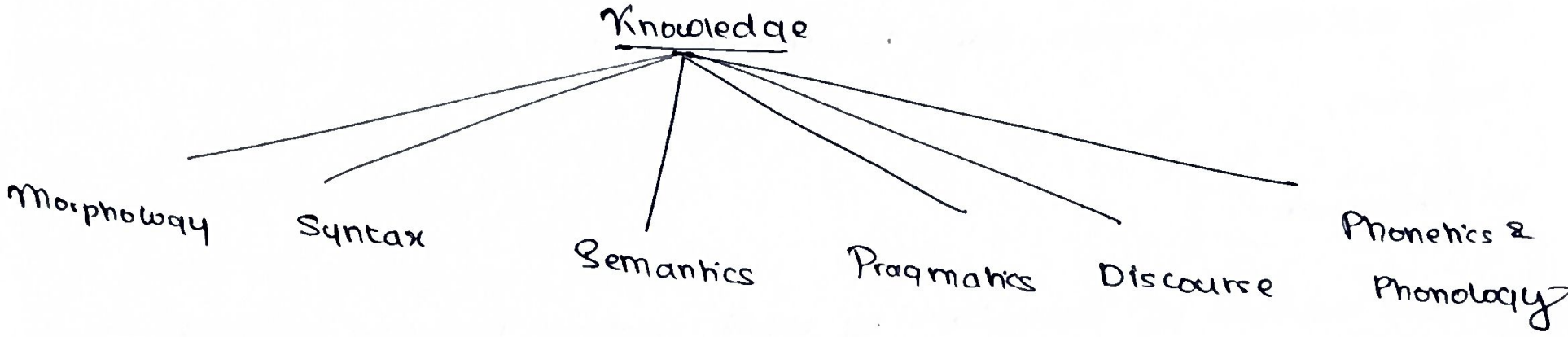
(v) LUNAR - question answering systems about moon rock

* Language and Knowledge

↓ medium of expression in which knowledge is deciphered

↓ a means of representation of content

* Knowledge Needed for NLP



A. Morphology

→ the branch of linguistics that studies patterns of word formation within and across languages, and attempts to formulate rules that model the knowledge of the speakers in those languages.

eg. dish, dishes, dishwasher.

B. Syntax

→ The branch of linguistics that studies the principles and rules for constructing sentences in natural languages.

Word Order - refers to the knowledge needed to order & group words

eg. (i) John hit Bill
(ii) Bill was hit by John
(iii) Bill, John hit

c. **Semantics**

→ refers to the study of meaning

→ can be of two types

a. lexical semantics - the meaning of all the words (individually)

b. compositional semantics - concerned with phrasal meanings, how individual words come together to make a broader, comprehensible sentence.

D. **Pragmatic**

→ the subfield of linguistics which studies the way in which context contributes to meaning.

→ It studies the kind of actions that speakers intend by their use of sentences (also called dialogue knowledge)

E- **Discourse**

eg. see the Eamore Railway set of Qs

→ Discourses are linguistic units composed of several sentences; in conversations, arguments or speeches

→ makes use of knowledge about how words like 'that' or pronouns like 'it' or 'she' refer to previous parts of the discourse.

eg: How many states were in the United States that year?
(Examine the earlier sentences that may have mentioned the year), also look at previously asked questions)

* Challenges of NLP

→ requires precise representation of content which is difficult due to ambiguity and vagueness of natural language.

→ inability to capture all the required knowledge - not possible to write procedures that imitate language processing as done by humans.

→ may be difficult to identify semantics like 9/11, the old man kicked the bucket

* Ambiguity

→ exists at almost every level

→ Input is said to be ambiguous if multiple, alternative linguistics structures can be built for it

eg. I made her duck

- made → can be verb or noun
- her → dative or possessive pronoun
- duck → semantically ambiguous - create or cook
- duck → syntactically ambiguous - can be transitive, ditransitive, action, direct object & verb

→ To decide whether duck is a noun or verb → use POS tagging

→ To decide whether make means create or cook → use word sense disambiguation

→ To decide whether her and duck are part of the same entity → use syntactic disambiguation

Resolution of POS tagging and word sense disambiguation is called Lexical disambiguation

* Regular Expressions

→ Regular expressions are used to explain text patterns.

→ It is a language for specifying text search strings.

→ It is used in every computer language, word processor and text processor tools like the Unix tools grep or Emacs.

→ Formally, a regular expression is an algebraic notation for characterizing a set of strings.

* Regular Expression Patterns

1. To look up a word = /word/

   eg. /woodchucks/ ⇒ There is a cute woodchuck.

2. Letters inside square brackets

   (i) To include both cases : [wW]oodchuck

   (ii) To include any digit : [0123456789]

   (iii) Match an upper case letter : [A-Z] - DrenchedBlossom

   (iv) Match a lower case letter : [a-z] - my beans

   (v) Match a single digit : [0-9] . Chapter 1 : down the rabbit hole

3. Negations - use a carat as the first symbol within square brackets

   [^ ...]

eg (i) not an uppercase : [^A-Z] - Hello!

(ii) neither s nor S : [^sS] - Snakes are cute.

(iii) neither e nor ^ : [^e^] - Mice

(iv) The pattern a^b : [a^b] = Look up a^b.

4. Disjunctions - for OR operation
   - use the | symbol

eg. woodchucks or groundhogs : [Ww]oodchucks|[Gg]roundhogs

use parantheses to make the disjunction apply only to a part of the word. [eg. pupp(y|ies)]

5. Optional previous character - ?

eg. To consider both spellings of the word color and neighborhood

colou?r ⇒ color or colour

neighbou?rhood ⇒ neighborbood or neighbourhood

6. Any character that can be substituted - .

eg. To consider all tense forms of the word begin

beg.n ⇒ begin, began, begun, even beg3n.

7. Kleene closure - * ⇒ zero or more of the previous character

eg. ooh*h ⇒ oh, ooh, oooh and so on

8. Positive closure - + ⇒ 1 or more of previous character

eg. o+h ⇒ oh, ooh, oooh...., baa+ ⇒ baa, baaa, baaaa ...

9. Anchors - Indicate what the line should start / end with.

^[ ] => line should start with what is in square brackets

$ => line should end with the character preceding the $ symbol

eg. (i) ^[A-Z] - a the San Francisco bridge — no match

- San Francisco

(ii) ^[^a-zA-Z] - "Hello"

(iii) .$ - can end with any character - hello , The end!

(iv) \.$ - should end with . - The end.

↙

\ is used as an escape sequence here, since just the . would mean retrieval of any character.

10. Blank spaces b vs B

\b matches the empty string at the beginning or end of a word

\B matches the empty string not at the beginning or end of a word.

eg. to match the word cat - \bcat\b would retrieve

I am a cat here

but not. I like cats

* RE Operator Precedence

Parantheses	( )
Counters	* + ? { }
Sequences and anchors	the ^my end $
Disjunction	|

example (i) /the*/ matches theeee but not thethe ;;

(ii) /cooky|lies/ matches cooky ~~lies~~ or ies but not cookies

(use parantheses to get that)

* Error Correction

→ Regex can be used to fix errors of 2 kinds:

(i) Match strings that should not have been matched

= False positives

(ii) Not matching strings that we should have matched

= False negatives

→ Reducing false positives means increasing accuracy or precision.

→ Reducing false negatives means increasing coverage or recall.

* Words

→ a unit of language that carries meaning and can stand alone or be combined with other words to form sentences

→ Punctuation is critical for finding boundaries of words and for identifying some aspects of meaning (?, !, " ")

→ also must identify real words when there are fragments, filled pauses (disfluencies)

Lemma - has the same stem & POS

refers to the base or dictionary form of a word

eg. lemma of running, runs, ran is run.

Wordform - refers to the full inflected or derived form of a word

eg. cat and cats are different wordforms

Type - no. of distinct words in the corpus

Token - an instance of that type in running text - usually individual words but also may be context-based (San Francisco should be a single token)

Eg they day back on the San Francisco grass and looked at the stars and their

15 (14) tokens

13 (12) types

* Number of words in a corpora

Let N = no. of tokens

V = Vocabulary

|V| = size of vocabulary

According to Heap's Law / Herdan's Law:

$$|V| = kN^{\beta}$$, where $0.67 < \beta < 0.75$

i.e the size of the vocabulary increases > square root of the no. of word tokens.

* Corpora

→ refers to a large collection of text documents or linguistic data that is used for statistical analysis / language modeling and other computational linguistics tasks.

→ The text may have:

(i) specific speaker(s) or writer(s)

(ii) a specific dialect of a specific language

(iii) specific time

(iv) specific place

(v) specific function

→ The corpora may have different languages, may be code-switched, may have different genres and author demographics

→ A corpus datasheet includes

(i) motivation - why, by whom, & funding

(ii) situation / context

(iii) means of collection, and the annotation process, language variety, demographics etc.

* Text Normalization

→ In the process of text normalization, three tasks are applied:

(i) tokenizing (segmenting) words

(ii) normalizing word formats

(iii) segmenting sentences

A: Tokenization

(1) Space-based tokenization

→ a very simple way to tokenize for languages that use space characters between words (Arabic, Cryllic, Greek, Latin etc) based writing systems

→ The UNIX `tr` command can also be used for this - given a text file, it outputs the word tokens and their frequencies.

Issues

(i) can't blindly remove punctuation (m.p.h, $45.45, #nlp, emails)

(ii) must handle clitics (words that don't stand on their own)

eg. we're, je in j'ai, le in l'honneur

(iii) must handle multiword expressions (MWE)

New York, rock 'n' roll

can use NLTK's regexp tokenizer

nltk.regexp_tokenize (text, pattern)

(iv) Language Issues

1. French - Is L'ensemble one token or two

Is it Le, L' or Le

L'ensemble = un ensemble.

2. German - german nouns are not segmented

IR in German needs a compound splitter.

3. Chinese & Japanese → no spaces between words

↓

may have multiple languages interminated

(Katakana, Hiragana, Kanji etc)

(2) Subword tokenization

→ Instead of white-space segmentation or single character segmentation : use the data to tell us how to tokenize

→ Subword tokenization recognizes that tokens can be part of words as well as whole words.

→ There are 3 common subword tokenization algorithms :

(i) Byte-Pair Encoding (BPE)

(ii) Unigram Language Modeling Tokenization

(iii) Wordpiece

→ All the subword tokenizers have 2 parts:

A. A token learner - that takes a raw training corpus and induces a vocabulary (a set of tokens)

B. A token segmenter that takes a raw test sentence and tokenizes it according to that vocabulary.

* Byte Pair Encoding (BPE) Learner

→ Let the vocabulary be the set of all individual characters

eg. {A, B, C, D ... , a, b, c, d ... }

→ Add a special end of word symbol '_' before spaces in the training corpus.

→ Repeat :

(i) choose the two symbols that are most frequently adjacent in the training corpus (say 'A', 'B')

(ii) add a newly merged symbol 'AB' to the vocabulary

(iii) Replace every adjacent 'A' 'B' in the corpus with AB

until k merges have been done.

Algorithm

```
function BYTE-PAIR ENCODING (strings C, number of merges k)
                                        returns vocab V
    V ← all unique characters in C
    for i=1 to k do
        tL, tR ← most frequent pair of adjacent tokens in C
        tNew ← tL + tR
        V ← V + tNew
    return V  replace each occurrence of tL, tR with tNew
```

Example Apply the BPE learner on the following corpus for three iterations.

Corpus : low low low low low lowest lowest newer newer newer newer newer newer wider wider wider new new

Ans

words & occurrences w/ end of word tokens

low _ - 5

lowest _ - 2

newer _ - 6

wider _ - 3

new _ - 2

Vocabulary:

_, l, o, w, e, s, t, n, r, i, d,

Iteration 1

Find most commonly occurring character pair

lo - 7

ow - 7

ne - 8

er - 9

Merge 'e', 'r' to er

add to vocabulary - _, l, o, w, e, s, t, n, r, i, d, er

Iteration 2

Find the most commonly occurring character pair

low - 7
ow - 7
ne - 8
er_ 9

merge 'er' '_' to 'er_'

add to vocabulary - , _, l, o, w, e, s, t, n, r, d, er, er_

Iteration 3

Find the most commonly occurring character pair

low - 7
ow - 7
ne - 8

merge 'n' 'e' to 'ne'

add to vocabulary - , _, l, o, w, e, s, t, n, r, d, er, er_, ne

* Byte Pair Encoding (BPE) Encoder

→ On the test data, run each merge learned from the trainingdata.

→ This is done (i) greedily

(ii) in the order the merges were learned

(iii) test frequencies don't play a role.

B. **Word Normalization**

→ refers to putting words / tokens in a standard format

(U.S.A or USA , am, is are be)

C. **Case Folding**

→ refers to different ways of handling case

Applications like IR - reduce all letters to lower-case

- only exception is when proper nouns are used mid-sentences (eg. General Motors)

Applications like sentiment analysis, MT, IE

- case is helpful (US vs. us is significant)

D. **Lemmatization**

→ reduce inflections or variant forms to base form

→ the correct dictionary headword form must be found.

eg. am, are, is → be

car, cars, car's → car

eg. the boy's cars are different colors

⇓

the boy car be different color

E. Morphology

→ The small meaningful units that make up words

Stem = core meaning-bearing units

Affixes = bits and pieces that adhere to stems - often with grammatical functions

F. Stemming

→ reduce terms to their stems in information retrieval

→ stemming is the crude chopping of affixes

→ language dependent

→ eg. automates, automatic, automation are all reduced to automat.

Porter Stemmer steps

Step 1a	sses → ss	caresses → caress
	ies → i	ponies → poni
	ss → ss	caress → caress
	s → ϕ	cats → cat
Step 1b	(*v*) ing → ϕ	walking → walk
	(*v*) ed → ϕ	plastered → plaster.

Step2

ational	→ ate	relational	→ relate
izer	→ ize	diainber	→ diainze
ator	→ ate	operator	→ operate

Step3

al	→ ϕ	revival	→ reviv
able	→ ϕ	adjustable	→ adjust
ate	→ ϕ	activate	→ activ

G. **Sentence Segmentation**

→ Punctuation marks !, ? for the end of sentences are unambiguous, but '.' is very ambiguous (esp. in abbreviations - Dr. & numbers 4.3)

→ A common algorithm for sentence segmentation is to tokenize first - use rules or a ML algorithm to classify a period as either

(i) part of the word

(ii) a sentence boundary

→ An abbreviation dictionary can also be used for this purpose.

* String Similarity and Minimum Edit Distance

→ String similarity helps find out the closest word to a given word.

- used in (i) spell correction

(ii) computational biology (to align sequences of nucleotides)

(iii) machine translation, information extraction, speech recognition

* Edit Distance

→ The minimum edit distance between two strings is the minimum number of editing operations, needed to transform one string to another with:

(i) insertion

(ii) deletion

(iii) substitution

Example - To compute the minimum edit distance, find the alignment.

```
I N T E * N T I O N
  E X E C U T I O N
  ↓ ↓ ↓ ↓
  d s s i s
```

→ If each operation costs 1 → distance = 5

each sub costs 2 → distance = 8

(Levenshtein Distance)

* Minimum Edit Distance Problem

→ Search for a path (a sequence of edits) from the start string to the final string.

Initial state: the word we're transforming

Operators: insert, delete, substitute

Goal state: the word we're trying to get to

Path cost: what we want to minimize - the no. of edits

* Minimum Edit Distance as a Search Problem

→ The space of all edit sequences is huge.

→ Naive navigation is not possible

→ Many distinct paths end up at the same state - don't keep track of all of them, just the shortest path to each of those revisited states.

* Minimum Edit Distance as a Dynamic Programming Problem

Dynamic programming: a tabular computation by combining solutions to solve a problem

For two strings:

X of length n

Y of length m

D(i,j) is defined as:

→ the edit distance between X[1...i] and Y[1..j]

→ the first i characters of X and the first j characters of Y

→ the edit distance between X and Y is thus D(n,m)

→ A bottom-up approach is used for MED:

- compute D(i,j) for small i, j
- compute larger D(i,j) based on previously computed small values.

i.e compute D(i,j) for all i (0<i<n) and j (0<j<m)

Algorithm (Levenshtein)

Initialization

$$D(i,0) = i$$
$$D(0,j) = j$$

Reccurence Relation

For each i = 1..M

For each j = 1..N

$$D(i,j) = \min \begin{cases} D(i-1,j)+1 \\ D(i,j-1)+1 \\ D(i-1,j-1)+ \begin{cases} 2 & \text{if } x(i) \neq y(j) \\ 0 & \text{if } x(i) = y(j) \end{cases} \end{cases}$$

Example1 Compute the MED between INTENTION and EXECUTION

N	9	8	9	10	11	12	11	10	9	8
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

min edit distance = 8

For Levenshtein

If chars match, copy diagonal value

else choose min of bottom +1, left +1 and diagonal value. +2

Example2 Compute the edit distance between Leda and deal when insertion cost = 1, deletion cost = 1 and substitution cost = 1

A	4	3	3	2	3
D	3	2	2	2	3
E	2	2	1	2	3
L	1	1	2	3	3
#	0	1	2	3	4
	#	D	E	A	L

* Backtrace for computing alignments

→ Just the edit distance isn't sufficient — sometimes characters to be aligned from both the strings

→ Do this by keeping a backtrace — each time one enter a cell remember where you came frame

→ Trace back the path from the upper right corner to get the alignment.

∴ The Recurrence relation can be rewritten as:

For each $i = 1 \ldots M$

For each $j = 1 \ldots N$

$$D(i,j) = \min \begin{cases} D(i-1,j) + 1 & \text{deletion} \\ D(i,j-1) + 1 & \text{insertion} \\ D(i-1,j-1) + 2 & \text{sub if } X(i) \neq Y(j) \\ 0 & X(i) = Y(j) \end{cases}$$

$$ptr(i,j) = \begin{cases} \text{LEFT} & \text{insertion} \\ \text{DOWN} & \text{deletion} \\ \text{DIAG} & \text{substitution} \end{cases}$$

* N-gram Language Models

* Language Models

→ Formal grammars (eg. regular & context-free) give a hard binary model of legal sentences in a language.

→ For NLP, a probabilistic model of a language that gives a probability that a string is a member of a language is more useful.

→ For a correct probability distribution, the probability of all sentences in a language must sum to 1.

→ Language models can be used for:

(i) speech recognition

(ii) OCR & handwriting recognition

(iii) Machine translation

(iv) generation

(v) context-sensitive spelling correction

(vi) completion & prediction

Probabilistic Language Model

goal - to compute the probability of a sentence or sequence of words

$$P(W) = P(w_1, w_2 \ldots w_n)$$

can also find probability of an upcoming word

$$P(w_5 \mid w_1, w_2, w_3)$$

→ A model that computes either of these:

$P(w)$ or $P(w_n | w_1, w_2 \ldots w_n)$ is called

a Language model.

* N-gram models

→ Estimate the probability of each word given prior context.

→ The number of parameters required grows exponentially with the number of words of prior context.

→ An n-gram model uses only N-1 words of prior context.

→ N-gram models decompose sentence probability into a product of conditional probabilities using the chain rule:

$$P(x_1, x_2 \ldots x_n) = P(x_1) \cdot P(x_2|x_1) \cdot P(x_3|x_1, x_2) \ldots P(x_n | x_1 \ldots x_n)$$

in general:

$$P(w_1, w_2 \ldots w_n) = \prod_i P(w_i | w_1 w_2 \ldots w_{i-1})$$

Example $P($" its water is so transparent"$) =$

$$P(\text{its}) \times P(\text{water} | \text{its}) \times P(\text{is} | \text{its water}) \times P(\text{so} | \text{its water is}) \times P(\text{transparent} | \text{its water is so})$$

The estimation of probabilities is done using the Markov Assumption

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

or $P(\text{the} \mid \text{transparent that})$

In general:

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

For a unigram model : $P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i)$

bigram model : $P(w_i \mid w_1 \ldots w_{i-1}) \approx P(w_i \mid w_{i-1})$

* Estimating bi-gram probabilities

use MLE = maximum likelihood estimate

$$P(w_i \mid w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

Example : Compute the probabilities for the following corpus using MLE

<s> I am Sam </s>

<s> Sam I am </s>

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

Ans: $P(I \mid <s>) = \frac{2}{3} = 0.67$

$P(am \mid I) = \frac{2}{3} = 0.67$

$P(Sam \mid am) = \frac{1}{2} = 0.5$

$P(</s> \mid Sam) = \frac{1}{2} = 0.5$

$P(Sam \mid <s>) = \frac{1}{3} = 0.33$

$P(do \mid I) = \frac{1}{3} = 0.33$

→ For a new sentence, find the product of the probabilities.

* Issues with bi-gram models

→ Multiplying the probability might cause a numerical underflow

To avoid this: sum their logs

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

→ Suffers from data sparseness

→ May have zero probabilities in bi-gram matrix, use smoothing techniques

* Smoothing Techniques

→ Refers to the task of re-evaluating zero-probability or low-probability n-grams and assigning them non-zero values

→ Some techniques include:

(i) add-one smoothing

(ii) good-turing smoothing

(iii) catching techniques

→ there's also

Kneser-key

Witten-bell

A. **Add-One Smoothing**

→ also called Laplace smoothing

→ Pretend that each seen one more time than we actually did, add 0 one to all the counts.

$$P_{\text{add-1 estimate}} : P(w_i|w_{i-1})_{\text{add-1}} = \frac{c(w_{i-1}, w_i) + 1}{c(w_{i-1}) + V}$$

B. **Good-Turing Smoothing**

→ use the count of things we've seen once to help estimate the count of things we've never seen.

N_c = count of things we've seen n-times

$$P_{GT} = c^* = \frac{(c+1)N_{c+1}}{N_c} \Big/ N$$

and

$$P^*_{GT}(\text{zero freq}) = \frac{N_1}{N}$$

Example : Consider that a series of fish that is caught is:

10 carp, 3 perch, 2 whitefish, 1 trout, 1 salmon, 1 eel

Ans. Find P(bass or catfish) and P(trout)

Ans (i) P(bass or catfish) = P_{GT}(zero seen)

$$= \frac{N_1}{N}$$

$N_1 = 3$ (1 trout, 1 salmon, 1 eel)

$N = 18$ (total fish)

$\therefore$ P(bass or catfish) = 3/18 = 1/6

(ii) $P(\text{trout}) = c^* = \frac{(c+1)\,N_{c+1}}{N_c}$

→ trout is seen once

→ c=1

$N_c = N_1 = 3$

$N_{c+1} = N_2 = 1$

$$c^* = \frac{2 \times 1}{3} = 2/3$$

$P^*_{GT}(\text{trout}) = 2/3 \,/ 18 = 1/27$

$\frac{N_1}{N}$

$\frac{(c+1)\,N_{c+1}}{N_c}$

ppt last q ans?

$P^*_{GT}(\text{trout}) = 1/27$

Example Consider the following corpus

<s> I am Sam </s>

<s> Sam I am </s>

<s> Iam Sam </s>

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

Using a bigram model with add-one smoothing, what is P(Sam|am) Include <s> and </s> as well.

Regex Example Questions

(1) Find all occurrences of the word "the" in a text or "The"

Ans [^a-zA-Z][tT]he [^a-zA-Z]

should not have any alphabets before or after 'the'

(2) A line containing only cat

Ans /^cat$/

??

(3) Beginnings of longer strings that start with 'un

Ans /\bun\B/

(4) A line containing The boat.

Ans /^The boat.$/

(5) Match 'There are 55 bottles of honey' but not 'There are 255 bottles of honey'.

Ans /\b55\b/

(6) Match 'Column 1 Column 2 Column 3' using the Kleene operator

/(Column_[0-9]+_)*/

column space any number, and this ~~any no. of spaces after column~~ repeats itself any no. of times

* Neural Language Model

→ an application of neural networks in NLP

Advantages over n-gram

→ can handle longer histories

→ can generalize over contexts of similar words

→ have higher predictive accuracy than n-gram language models

Disadvantages over n-gram

→ slower to train

Applications

→ machine translation

→ dialogue systems

→ language generation

* Neural Language model architectures

→ GPT-3

→ XLNet -

→ OPT

→ BLOOM

→ FFNN

* Feed Forward Neural Language Model

Input : representation of some number of previous words

Output: probability distribution over possible next words

Goal : Approximate the probability of a word given the entire prior context.

→ Neural language models represent words from the prior context by their embeddings rather than just by their word identities

→ Using embeddings allows neural language models to generalize better to unseen data

eg: If the training data is:

The dog needs to be fed

and the testing data is:

I need to make sure that the cat gets _____.

→ A FFNN would be able to predict that the next word is fed, while an n-gram model would not, since it has never seen the word cat before.

→ A FFNN's embeddings help understand that cat & dog are similar.

Algorithm

Step1: Generate embeddings

1. Start with a one-hot vector for each word in the vocabulary
2. Random initialize the initial weights
3. Maintain a separate vector of weights for each vocabulary word.

4. Concatenate the embeddings for the context words to produce the embedding layer e.

$$e = (Ex_1, Ex_2, \ldots Ex_n)$$

Step 2: Calculate hidden layer weights

1. Multiply by W and add b and pass through a ReLU activation function to get the hidden layer h.

$$h = \sigma(We + b)$$

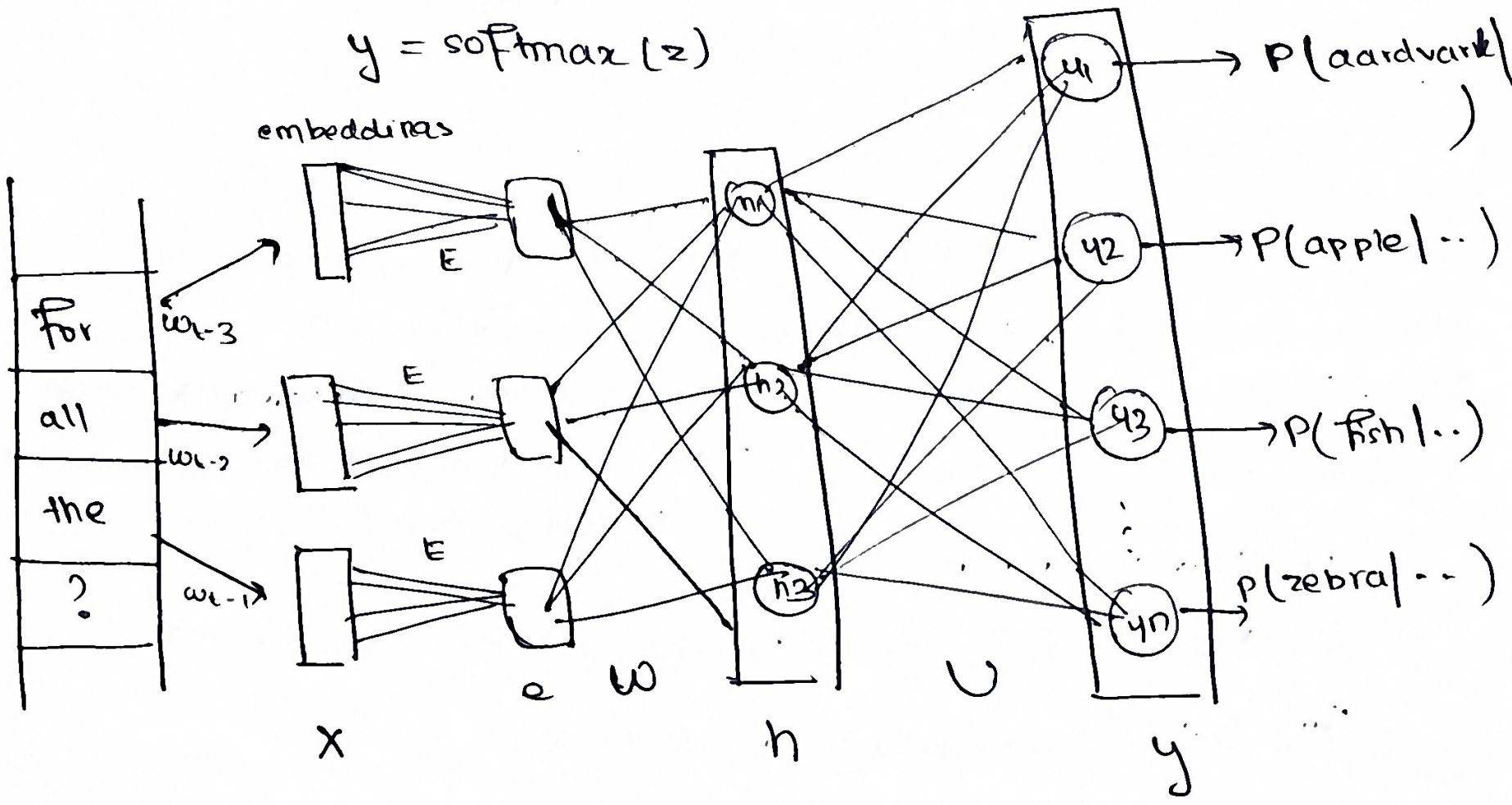
Step 3

1. Multiply by U (hU)

$$z = Uh$$

Step 4 - Apply softmax - each node estimates the probability of the next word.

$$y = \text{softmax}(z)$$



* Recurrent Neural Network

→ used when data is sequential like time-series data or text data.

→ RNNs are a type of neural network where the output from the previous step is fed as input to the current step.

* Issues with FF NN

(i) cannot handle sequential data

(ii) considers only the current input

(iii) cannot memorize previous inputs

*Applications of RNN

(i) Language modeling

(ii) POS Tagging

(iii) Sequence classification

* Key Features of RNNs

→ has a built-in capacity to handle temporal information, can handle info right from the start of a sequence.

→ can accept variable length inputs without the use of fixed sized windows.

→ The value of an input is dependent upon outputs from previous timesteps, done with loops which allow info to persist over time. Info is stored between timesteps using an internal hidden state, and fed back into the model the next time an input is read.

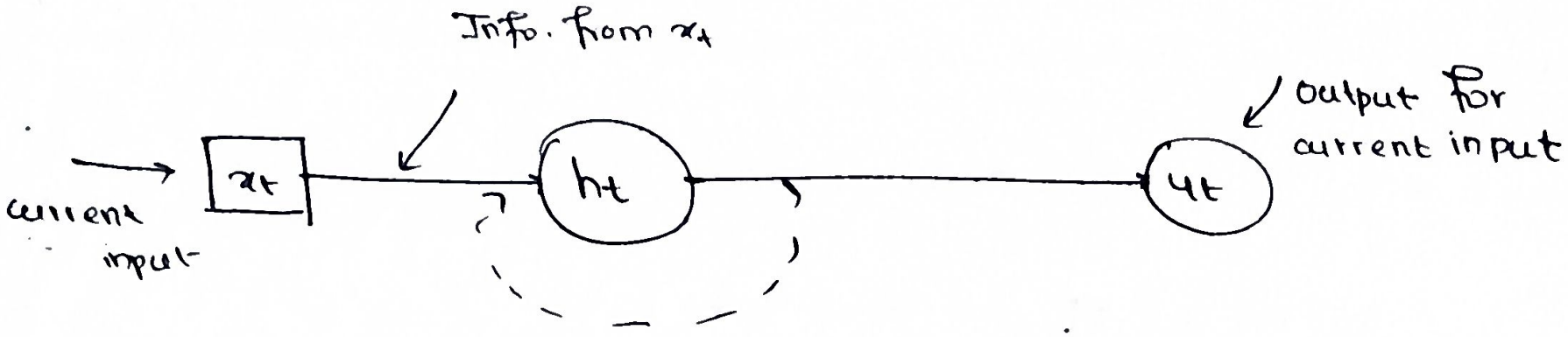
→ Some RNN varieties are:

(i) Vanilla RNNs

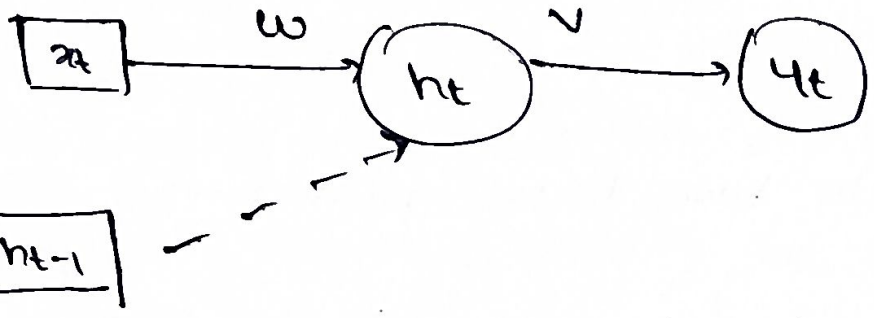
(ii) Long Short Term Memory Networks (LSTM)

(iii) Gated Recurrent Units (GRUs)

* Vanilla RNN Architecture



The unrolled view would be:



* Formal equations for RNNs

→ For a regular FFNN, the equations are:

$$h = \sigma(Wx + b)$$
$$z = Uh$$
$$y = \text{softmax}(z)$$

→ In RNNs, add the weights X activation values from the previous steps

$$h = \sigma(Wx + Uh_{t-1} + b)$$
$$z = Vh_t$$
$$y = \text{softmax}(z)$$

W, U, V are shared across all timesteps

* Formal Algorithm

$h_0 \leftarrow 0$ Initialize initial activations from the hidden layer, h0

for $i \leftarrow 1$ to len(x) do:

$$h_i \leftarrow g(Uh_{i-1} + Wx_i + b)$$

$$y_i \leftarrow f(Vh_i)$$

* Training RNNs

→ The core elements like the loss fn, optimization fn, backpropagation are still present

→ An additional set of weights have to be updated - i.e weights from hidden layer t-1 to current hidden layer at t

Forward Inference

(i) Compute h_t and y_t at each timestep.

(ii) Compute the loss at each timestep

Backward Inference

(i) Process the sequence in reverse

(ii) Compute the required error gradients each step backwards in time.

Weight updation

→ There are 3 sets of weights that have to be updated:

(i) W = weights from input layer to hidden layer.

(ii) U = weights from the previous hidden layer to the current hidden layer.

(iii) V = weights from the hidden layer to the output.

* Recurrent Neural Language Model

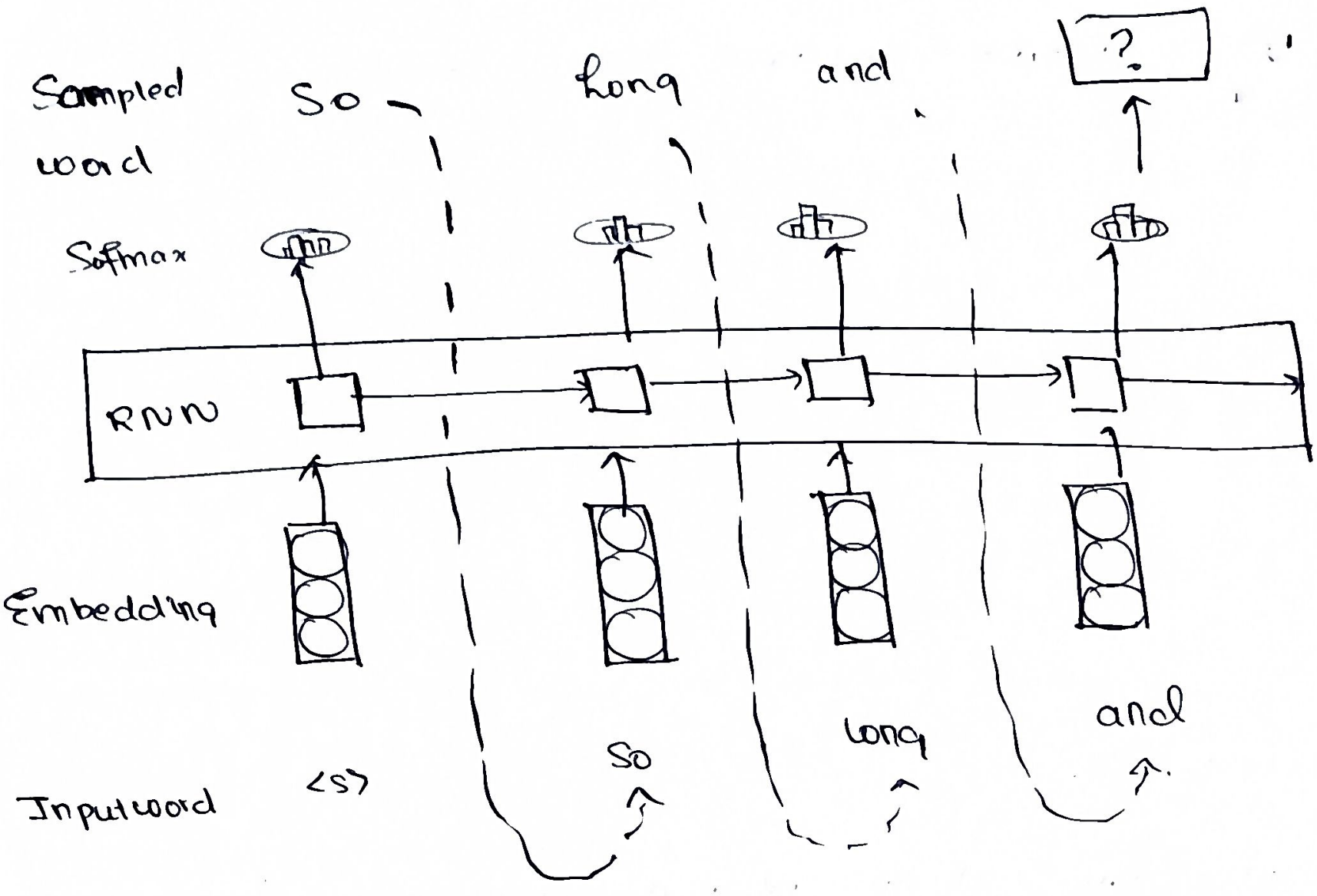
→ They process sequences one word at a time - They avoid constraining the context size.

→ The hidden state embodies information about all the preceding words, all the way back to the beginning of the sequence.

Autoregressive Generation Algorithm

1. Sample the 1st word in the output from the softmax distribution that results from using <s> as input.
2. Retrieve an embedding
3. Use as input in the next timestep.

   Combine the weighted sums of (a) input embeddings
   (b) activations of hidden layer from previous timestep.
4. Generate a set of o/ps from activations in hidden layer.
5. Pass outputs through softmax and repeat for the next words until </s> is read.


Sampled word
So
long
and
?
Softmax
RNN
Embedding
Input word
<s>
So
long
and