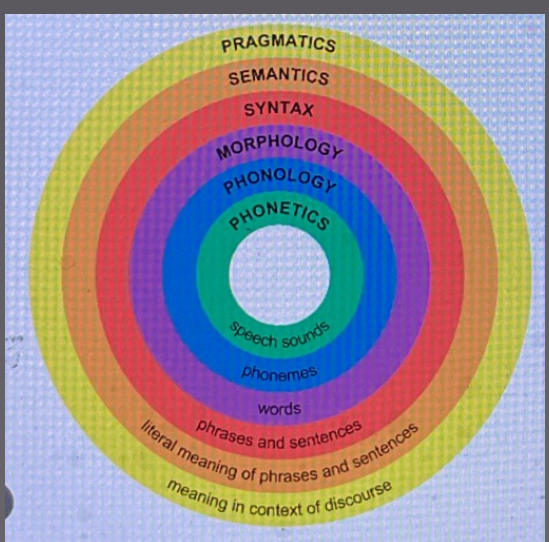
LLMs

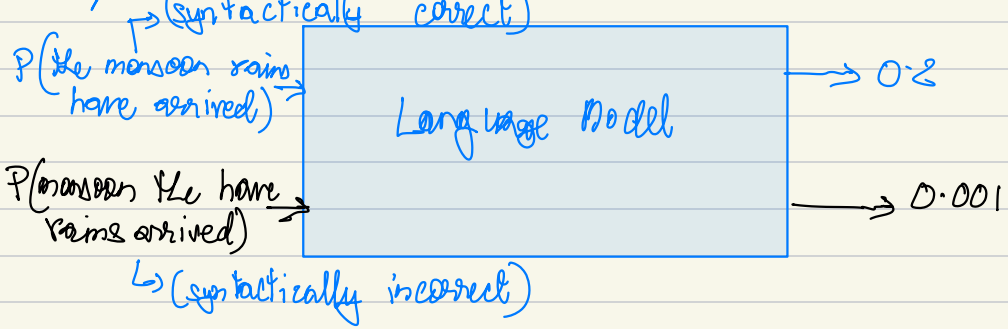
LCS2 IIT Delhi



Lecture 1 – Introduction to Language Models

What is a language model (LM)?

* Language model gives the probability distribution over a sequence of tokens.



* Similarly, it should also be able to understand semantics as well.

eg: The mouse ate the cheese (syntactically & semantically correct)
The cheese ate the mouse (syntactically correct, semantically wrong)

Vocabulary
↳ a dictionary which contains all the unique tokens

Vocabulary

V = {arrived, delhi, have, monsoon, rains, the}

Language models can 'generate' text

* Consider a sequence of tokens $\{x_1, x_2 \ldots x_L\}$ where $x_1, x_2 \ldots x_L$ are in vocabulary V

Notation: $P(x_1, x_2 \ldots x_L) = P(x_{1:L})$

* Using chain rule of probability:

$$P(x_{1:L}) = P(x_1) \cdot P(x_2|x_1) \cdot P(x_3|x_1, x_2) \ldots P(x_L|x_1, x_2 \ldots x_{L-1})$$

$$= \prod_{i=1}^{L} P(x_i | x_{1:L-1})$$

Q) So what does the term $P(x_3|x_1, x_2)$ mean?

A) Given the input 'the monsoon', an LM can calculate $P(x_i|$'the monsoon'$) \; \forall x_i \in V$

How?

- We assume that we've already computed a distribution
- Given the context, the task is to predict the next word given a distribution
- The distribution can be obtained from the corpus itself

for generation, we sample a token from this probability distribution

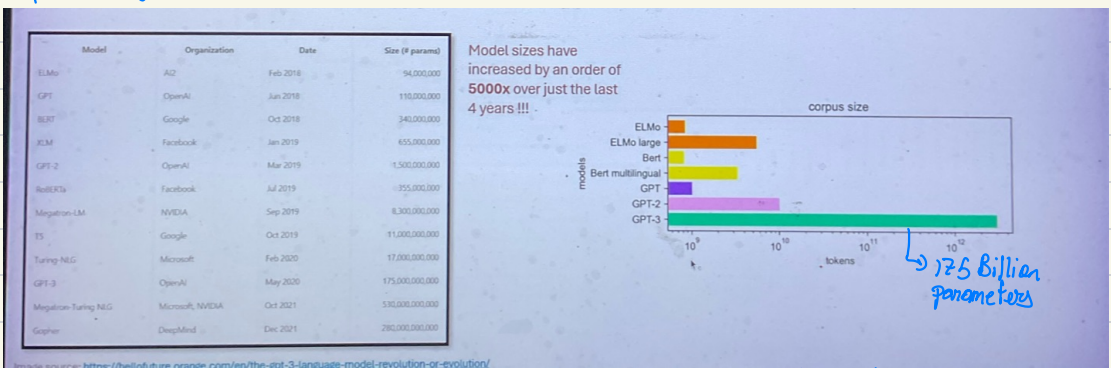
$$x_i \sim P(x_i | x_{1:i-1})$$

→ Autoregressive LMs calculate this distribution efficiently, e.g. using 'deep' neural networks

'Large' Language Models

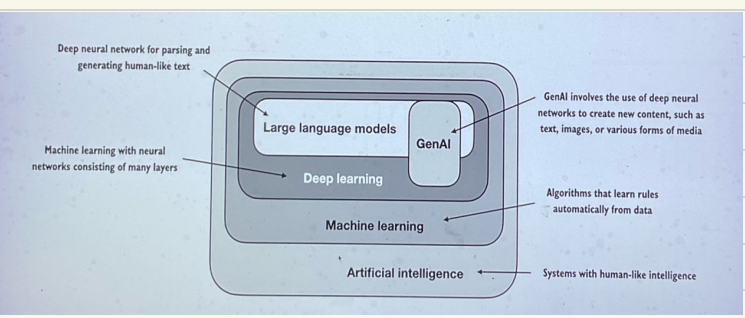
* The 'Large' in terms of 'model size' and 'size of the dataset'

Model	Organization	Date	Size (# params)
ELMo	AI2	Feb 2018	94,000,000
GPT	OpenAI	Jun 2018	110,000,000
BERT	Google	Oct 2018	340,000,000
XLM	Facebook	Jan 2019	655,000,000
GPT-2	OpenAI	Mar 2019	1,500,000,000
RoBERTa	Facebook	Jul 2019	355,000,000
Megatron-LM	NVIDIA	Sep 2019	8,300,000,000
T5	Google	Oct 2019	11,000,000,000
Turing-NLG	Microsoft	Feb 2020	17,000,000,000
GPT-3	OpenAI	May 2020	175,000,000,000
Megatron-Turing NLG	Microsoft, NVIDIA	Oct 2021	530,000,000,000
Gopher	DeepMind	Dec 2021	280,000,000,000

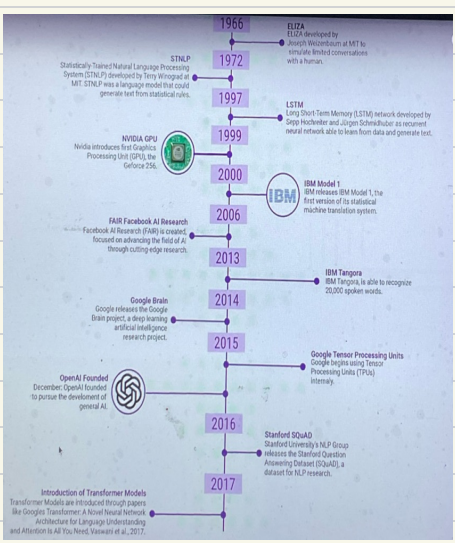


Other recent models : PaLM (540B), OPT (175B), BLOOM (176B)
Gemini-Ultra (1.56T - conjecture), GPT-4 (1.76T - conjecture)

LLMs in AI landscape



Evolution of LLMs



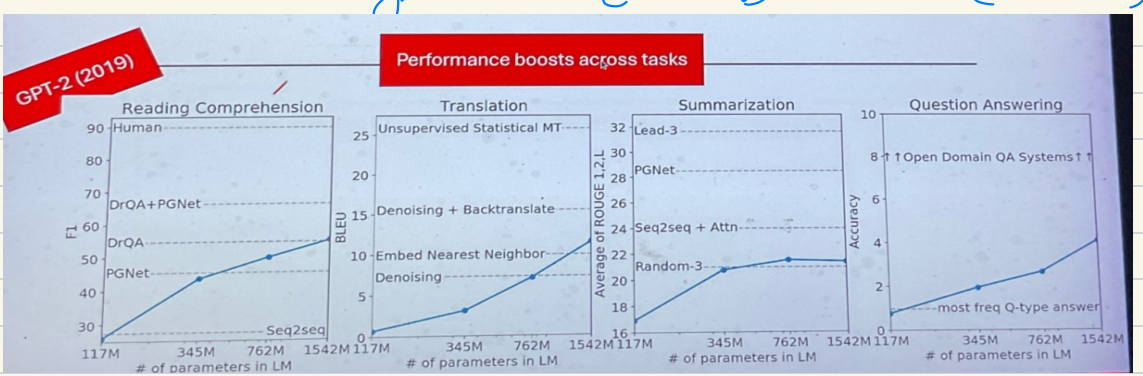
Post Transformers Era

Transformers – 2017

BERT → 2018
- → Introduced Masked Language Modeling
- → Achieved SOTA on 11 NLP tasks

GPT → 2018
- → Decoder only model by Open AI

GPT 2 → 2019
- → 117M → 1.5B parameters
- → Context length : GPT-1 (512 tokens) GPT-2 (1024 tokens)



T5 → 2019 (by Google)
- → Text-to-Text transformer
- → Pre-training similar to BERT
- → Encoder-Decoder model

RoBERTa → Replicatn study of BERT pretraining (2019)
- → Found that BERT was significantly undertrained & could outperform later models

XLM → 2019
- Multilingual model

GPT-3 → 2020
- 175B parameters
- In context learning
- No longer open source (After GPT 1 & GPT 2)

PaLM → 2022
- 540B params
- for conversational AI
- Not open source

OPT → 2022
- Open Pretrained Transformer (By META)
- A suite of decoder-only pre-trained transformers ranging from 125M to 175B params
- Open source

Nov 30, 2022 → Chat GPT introduced

Bard – By Google (2023)

Llama – By META (2023) – family of open source models

Claude – By Anthropic (2023)
- Start-up by ex OPEN AI founders

GPT-4 → Mar 2023
- Model size not known

Mistral 7B → By Mistral AI
↳ focus on smaller model

Grok → By xAI (Nov 2023)

Gemini → By Google (Dec 2023)

2024 → Gemma 2
↳ GPT 4o, GPT 4-o1
↳ Llama 3
↳ Codestral

Why LLMs ?

* LLMs show emergent capabilities, not previously observed in small LMs
- In-context learning: A pre trained language model can be guided with only prompts to perform different tasks (w/o seperate task specific fine tuning)
↳ In context learning is an example of emergent behavior.

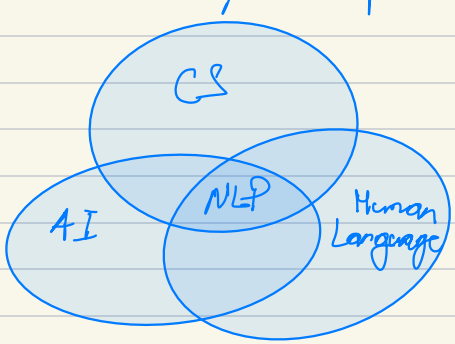
* LLMs are widely adopted in real world
Research: LLMs have transformed NLP research & achieved SOTA across a wide range of tasks like sentiment classificatn, question answering, summarisatn & machine translatn
Industry: Many models are now used in productn environments eg: BERT, XLM, GPT-3.5, 4

* LLMs also have a lot of risk
  - Reliability & Disinformation: Hallucination is a problem in critical tasks
  - Social bias: Predictions show disparity across demographic groups & their predictions can enforce stereotypes
  - Toxicity: LLMs can generate toxic/hateful content
  - Security: from data poisoning

Lecture 2 — Introduction to NLP

Q) What is Natural language?
A) Evolved naturally through repitition & use without conscious planning

Q) What is NLP?
A) A field of computer science, AI & computational linguistics concerned with the interaction b/w computers & human (natural) language



Why is this interesting?

More than 6500 official languages in the world (not counting dialects)

"Computing Machinery & Intelligence" proposed what is now known as the Turing Test

Setup

→ 2 rooms, 2 humans & a computer
Room 1: One human C
Room 2: One computer (A) & one human (B)
→ Response generated from room 2 (either by A or B)
→ C has to figure out the source of the response
- If C is successful → "A" failed the Turing Test
- Else → "A" passed the Turing Test

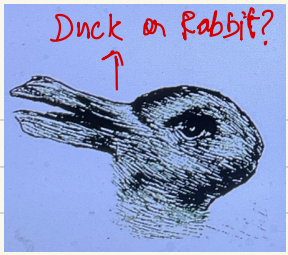
*) 1957 - Noam Chomsky's Syntactic Structures revolutionised linguistics with 'universal grammar' a rule-based system of syntactic structures

Why is NLP challenging?

Ambiguity

*) How might think the statement is factually incorrect or that Trumps should be Trump's but the fact is that Trumps was used a verb here (like beat, outdo or surpass)
Thus, the sentence is correct

Similarly, this sentence:
↳ Virat Kohli was on fire last night. He destroyed the opponent team.



But ambiguity is present in all dimensions.

Ambiguity in language

*) I saw a girl with a telescope. (Who had the telescope?)

But, I saw a girl with a bicycle } – No ambiguity!
I saw a boy with a telescope }

*) Mary had a little lamb.
↳ Mary possesses a lamb
or
↳ Mary ate a lamb

*) Mujhe aapko mithai khilani padegi

Kon kisko mithai khilayega

मुझे आपको मिठाई खिलानी पड़ेगी
↳ कौन किसको मिठाई खिलाएगा

Pragmatics → Identifying the meaning of sentence based on the context (tone, intonation, movement, etc.)

*) I ate rice with
- → spoon (tool)
- → curd (auxiliary food)
- → Ankur (a company/person)

Same Syntax, different semantics

*) Lets eat Grandma } – Punctuation is important
Lets eat, Grandma }

*) A woman, without her man, is nothing
A woman: without her, man is nothing

Buffalo buffalo Buffalo buffalo buffalo buffalo Buffalo buffalo → Valid sentence

*) Buffalo (3 meanings)
- Noun: Animal (plural is also buffalo)
- Proper Noun: US State
- Verb: To bully someone

Buffalo buffalo, whom other Buffalo *buffalo* buffalo, buffalo Buffalo buffalo

Dmitri Borgmann's *Beyond Language: Adventures i*

Meaning: buffalo of Buffalo state, whom other buffalo of Buffalo State bullied, bullied another buffalo of Buffalo state

Why else is NLP difficult?

Non-standard English

Great job @justinbieber! Were SOO PROUD of what youve accomplished! U taught us 2 #neversaynever & you yourself should never give up either ♥

Segmentation Issues

the New York-New Haven Railroad

the New York-New Haven Railroad

Idioms / Multiword

dark horse
get cold feet
lose face
throw in the towel
Khana-wana (Echo)

Neologisms → not in dictionary

unfriend
Retweet
bromance

World Knowledge

Mary and Juhi are sisters.
Mary and Juhi are mothers.

Tricky Entity Names

Where is *A Bug's Life* playing ...
Let It Be was recorded ...
... a mutation on the *for* gene ...

A single token - a song

NLP

Natural Language Understanding

Natural Language Generation

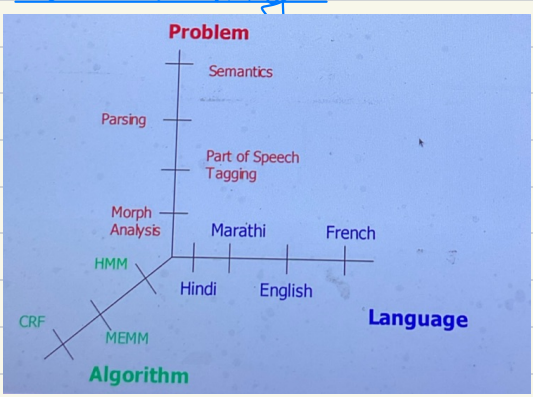
NLP Layers

*) Understanding the semantics is a non-trivial task.
*) Needs to perform a series of incremental tasks to achieve this.
*) NLP happens in layers

Pragmatics & Discourse	Study of semantics in context.
Semantics	Meaning of the sentence.
Parsing	Syntactic structure of the sentence.
Chunking	Grouping of meaningful phrases.
Part of speech tagging	Grammatical classes.
Morphology	Study of word structure.

Increasing Complexity Of Processing

NLP Trinity



The algorithms on the Z-axis are used to solve the problems on the Y-axis for the languages on the X-axis

MMM - Hidden Markov Models
CRF - Conditional Random fields
MEMM - Max. Entropy Markov Models

Word & Token

* phoneme - smallest unit of sound in a language that can change the meaning of a word.

Word: Smallest sequence of phonemes* of a spoken language that can be uttered in isolation.

* grapheme - smallest unit of writing in a language that represents a sound (phoneme)

Word Segmentat^{n}/Tokenisat^{n} : Break a string of characters into a sequence of words

Token: Smallest sequence of graphemes* that are delimited with some predefined characters (space, comma, full stop etc)

Ram, Shyam, and Mohan are playing.	⇒	[Ram] [,] [Shyam] [,] [and] [Mohan] [are] [playing] [.]
21,53,010 COVID cases in India.	⇒	[21] [,] [53] [,] [010] [COVID] [cases] [in] [India] [.]
		[21,53,010] [COVID] [cases] [in] [India] [.] ✓
Check this out...https://www.abc.com	⇒	[Check] [this] [out] [.] [.] [.] [https] [:] [/] [/] [www] [.] [abc] [.] [com]
		[Check] [this] [out] [...] [https://www.abc.com] ✓
#GreatDayEver	⇒	[#] [Great] [Day] [Ever]

Some tokenisat^{n} (using comma) cannot be used because the number is one token

↳ full stop cannot be used as delimiter here

↳ No space between words

Morphology

*) Field of linguistics that studies the internal structure of words
- ↳ How they're formed
- ↳ Relationship to other words in the same language

> **Morphology** is the study of words, how they are formed, and their relationship to other words in the same language. It analyzes the structure of words and parts of words, such as stems, root words, prefixes, and suffixes.

*) It defines word format^{n} rule from the root word.

*) Morpheme is the smallest linguistic unit that has semantic meaning

Ex: "Pre", "ed", "ing", "s", "es", etc.
- ↳ Dogs ⇒ dog + s (plural)
- ↳ Going ⇒ go + ing (present participle)
- ↳ Independently ⇒ independent + ly (Adverb)
  - in + dependent + ly (Negation)
  - in + depend + ent + ly (relying)
  - in + depend* + ent + ly

* pend: (verb) to remain undecided or unsettled

* How to do this? → finite State Automata, etc.
Morphology seperates the root word from prefixes, suffixes, etc.

Stemming
Lemmatisation } – 2 algorithms for morphological analysis

* Stemming → Returns the root word (may not be a dictionary word)
  - → Very fast
  - → 2 words from the same lemma will have the same stemmed word

* Lemmatisation → Returns the root word (always a dictionary word)
  - → Slow, requires a dictionary

* English, Chinese, etc → morphologically poor
Hindi, Turkish, Hungarian → morphologically rich

English	Hindi	Linguistic property
I will go.	मैं जाऊँगा।	Different morphological forms of word 'will go' in Hindi
We will go.	हम जाएंगे।	
You will go.	तुम जाओगे।	
He will go.	वह जाएगा।	
She will go.	वह जाएगी।	

verb has gender

Syntax

Syntax concerns the way in which words can be combined together to form (grammatical) sentences.

Pragmatics & Discourse	Study of semantics in context.
Semantics	Meaning of the sentence.
Parsing	Syntactic structure of the sentence.
Chunking	Grouping of meaningful phrases.
Part of speech tagging	Grammatical classes.
Morphology	Study of word structure.

Increasing Complexity Of Processing

Parts-of-speech (POS)

*) Grammatical class of the word

He	ate	an	apple
PRP	VBD	DT	NN

Tags

PRP: Personal Pronoun
VBD: Verb, Past
DT: Determiner
NN: Noun, Singular, Mass
TO: *to*
IN: Preposition

- 45 tags in Penn Treebank tagset
- 146 tags in C7

*) PoS disambiguation:
A word can belong different grammatical classes

He	went	to	the	park	in	a	car	.
PRP	VBD	TO	DT	NN	IN	DT	NN	.

They	went	to	park	the	car	in	the	shed	.
PRP	VBD	TO	VB	DT	NN	IN	DT	NN	.

Chunking

*) Identificatⁿ of non-recursive phrases (noun, verb, etc.)

eg: He went to the Indian city Mumbai
↓ ↓ ↓ ↓
NP VP PP NP

Mumbai green lights women icons on traffic signals
↓ ↓ ↓ ↓ ↓
NP PP NP VP NP

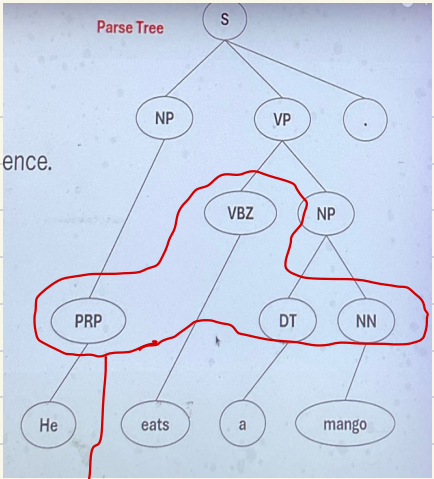
NP - Noun phrase
VP - Verb phrase
PP - Prepositⁿ phrase

Parsing (Syntax processing)

*) Validate the grammatical structure of the sentence.

* Let vocabulary: [the, mango, he, eats. . .]
He eats a mango ✓
He mango eats a ✗

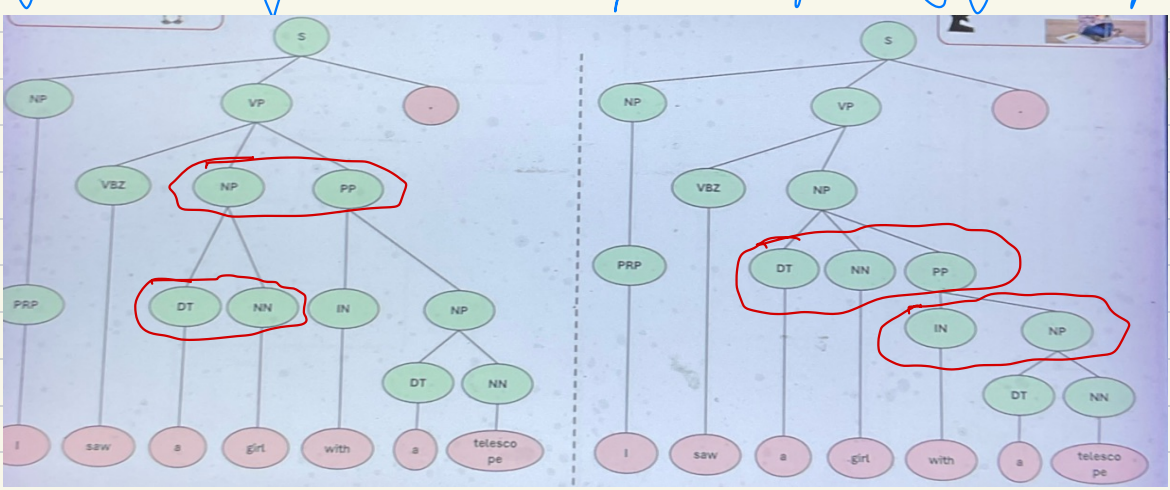
* The sequence of words must follow the grammatical structure of the language to form a valid sentence.
  - Construct a parse tree



↳ Task is to group 1st level POS tags into chunks

Syntactic Ambiguity

eg: I saw a girl with a telescope (same previously given example)



Semantics

Semantics & Pragmatics are the glue that connect language to the real world

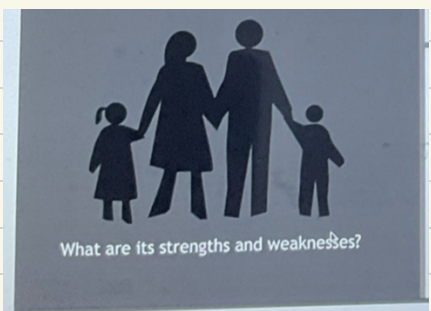
* Semantics is concerned with the meaning of the word & how to combine words into meaningful phrases & sentences.

- Decompositional – what the "components" of meaning ("in") a word are
- Ontological – How the meaning of word relates to the meanings of other words.
- Distributional – what context the word is found in relative to other words

Decompositional Semantics

*Decompositional semantics divides the meaning of words into components

boy [+human, -female, -adult]	girl [+human, +female, -adult]
man [+human, -female, +adult]	woman [+human, +female, +adult]

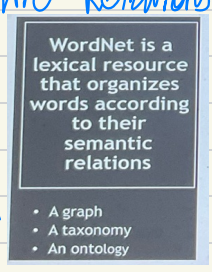


Ontological Semantics

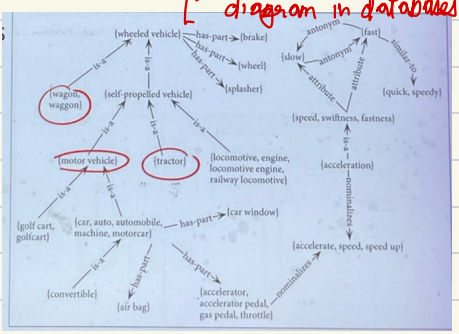
* Ontological semantics says that the meaning of a word is its relationship to other words

The basic (ontological) Semantic Relations

- Synonymy - equivalence
  - <small, little>
- Antonymy - opposition
  - <small, large>
- Hyponymy - subset; is-a relation
  - <dog, mammal>



→ similar to E.R. diagram in databases



- Hypernymy - superset - <animal, dog>
- Meronymy - part-of relation - <liver, body>
- Holonymy - has a relation - <body, liver>

Distributional Semantics → the premise behind all recent embedding models like Word2Vec, GloVe, etc.

> The meaning of words can be derived from their distributional properties in large corpora of text. It relies on context in which word appears

Example: The meaning of the word "cat" can be inferred from the contexts it appears in, such as sentences where it co-occurs with words like "pet", "animal", "meow", "feline"

The co-occurrence matrix

	leash	walk	run	owner	pet	bark
dog	3	5	2	5	3	2
cat	0	3	3	2	3	0
lion	0	3	2	0	1	0
light	0	0	0	0	0	0
bark	1	0	0	2	1	0
car	0	0	1	3	0	0

↳ the no of times the words co-appear together

cat co-appears with pet & owner like dog but not with the word "bark"
↳ still quite semantically similar

dog & car on the other hand do-not co-appear with many similar words

Similarity can be calculated by considering each row as a vector & calculating cosine similarity, dot product, etc.

Pragmatics

* Pragmatics considers [Thomas, 1995]:
  - the negotiation of meaning b/w speaker & listener.
  - the context of utterance
  - the intention of the user

Context/world knowledge: Employee late to office

Utterance: Do you know what time it is?
Literal Meaning: Are you aware of the current time?
Pragmatic Meaning: Why are you so late?

Intention

Utterance: Can you pass the water bottle?
Literal Meaning: Are you able to pass the water bottle?
Pragmatic Meaning: Pass me the water bottle.

Discourse

* Processing of sequence of sentences.

Mother to John: Go to school. Its open today. Do you plan to bunk? father will be very angry.

Discourse processing helps answer the questions:
- what is open?
- Bunk what?
- Why will father be angry?

Tasks of Interest

Semantic Role Labelling (SRL)

* Identify the semantic role of each argument (noun phrase) w.r.t the predicate (main verb) of the sentence.

John	drove	Mary	from	Delhi	to	Pune	in	his	car
Agent		Patient		source		destination			instrument

Ram	hit	Shyam	with	a	hockey	stick	yesterday
Agent		Patient			instrument		time

↳ Roles are already given. Task is to assign the role to the words.

Textual Entailment (TE)

* Determine whether a natural language sentence entails (implies) another under an ordinary interpretation.

(*Ram hit Shyam with a hockey stick yesterday.* → *Shyam got hurt*)	⇒ Positive TE
(*Ram hit Shyam with a hockey stick yesterday.* → *Shyam did not get hurt*)	⇒ Negative TE
(*Ram hit Shyam with a hockey stick yesterday.* → *Shyam got his first goal*)	⇒ Non TE

Co-reference Resolution

* Two refering expressions used to refer to the same entity are said to 'co-refer'.
* Determine which phrases in a document co-refer.

eg: John shows Bob his (→ John or Bob?) Toyota yesterday. It's similar to the one I bought 5 years ago.

That (Is this the current one or the one that was bought 5 years ago) was really nice, but he (→ John or Bob) likes his one even better.

Information Extraction

* Extracting relevant piece of information.

- Named Entity Recognition (NER):
  - Identify names (Proper nouns)
    - [India]$_{Location}$ born [Sundar Pichai]$_{Person}$ is the CEO of [Google]$_{Organization}$ and its parent company [Alphabet]$_{Organization}$
- Relation Extraction:
  - Relation among entities
    - CEO(Sundar Pichai, Google), CEO(Sundar Pichai, Alphabet), Born-at(Sundar Pichai, India), ParentOrg(Alphabet, Google)

Word Sense Disambiguation (WSD)

*) What does a word mean?

This fisherman went to the bank.
↳ financial bank or river bank?

- The fisherman went to the bank to withdraw money.
- The fisherman went to the bank to fish.

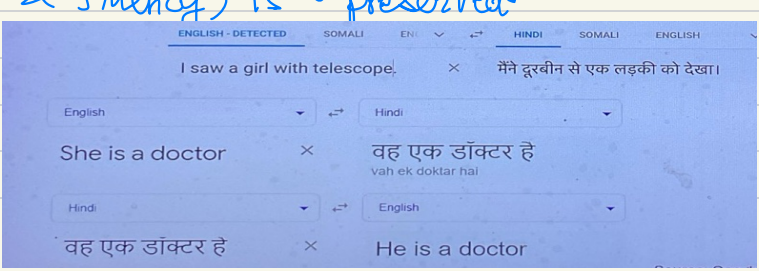
Sentiment Analysis

*) Extract polarity orientation of the subjectivity.

- Really superb pillow. Love to sleep on it.. very comfortable... ⇒ Positive
- It's a mass Chinese product. Too expensive. Thin and useless ⇒ Negative
- My neighbours are home and it's good to wake up at 3am in the morning. ⇒ Negative?
- Campus has deadly snakes. ⇒ Negative
- Shane Warne is a deadly spinner. ⇒ Positive?
- The food was cheap. ⇒ Positive?
- Not to mention the cheap service I got at the restaurant. ⇒ Negative
- Movie was 4 hours long. ⇒ Neutral?

Machine Translation

*) Given a sentence in the source language L1, convert it to target language L2, such that the semantic (adequacy & fluency) is preserved.



Summarisation

*) Given a document, summarize the semantics (extract relevant information) in shorter length text.

3 types → Extractive (copy, paste)
→ Abstractive (identify, understand & rephrase)
→ Aspect based (only summarize w.r.t to a particular aspect)

Question Answering

*) Answer natural language questions based on information present in repository.

Factoid Questions

- Question: Who is the author of the book Wings of Fire?
- Answer: A. P. J. Abdul Kalam

List Questions

- Question: What are the islands in India?
- Answer: Andaman Island, Nicobar Island, Labyrinth Island, Barren Island

Descriptive Questions Difficult

- Question: What is Greenhouse effect?
- Answer: The analogy used to describe the ability of gases in the atmosphere to absorb heat from the earth's surface.

Dialog Systems & Chatbots

C_1:	...I need to travel in May.
A_1:	And, what day in May did you want to travel?
C_2:	OK uh I need to be there for a meeting that's from the 12th to the 15th.
A_2:	And you're flying into what city?
C_3:	Seattle.
A_3:	And what time would you like to leave Pittsburgh?
C_4:	Uh hmm I don't think there's many options for non-stop.
A_4:	Right. There's three non-stops today.
C_5:	What are they?
A_5:	The first one departs PGH at 10:00am arrives Seattle at 12:05 their time. The second flight departs PGH at 5:55pm, arrives Seattle at 8pm. And the last flight departs PGH at 8:15pm arrives Seattle at 10:28pm.
C_6:	OK I'll take the 5ish flight on the night before on the 11th.
A_6:	On the 11th? OK. Departing at 5:55pm arrives Seattle at 8pm, U.S. Air flight 115.
C_7:	OK.

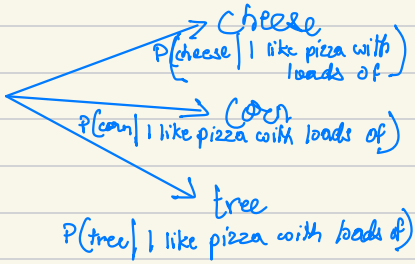
Lecture 3.1 – Introduction to Language Models

Introduction to Statistical Language Models

Next word Prediction

Guess the next word in the sequence...

I like pizza with loads of _____.



$P(\text{cheese} \mid \text{I like pizza with loads of}) > P(\text{corn} \mid \text{I like pizza with loads of}) \gg P(\text{tree} \mid \text{I like pizza with loads of})$

Entropy: A measure of randomness

$$H(x) = -\sum p(x) \log_2 p(x) \approx 4.2 \text{ (for the occurrence of character 'x')}$$

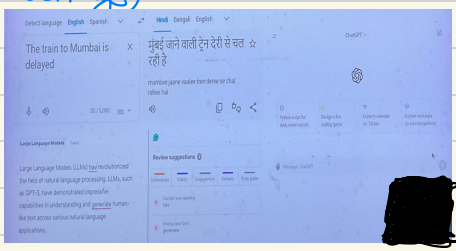
$x = \frac{1}{27}$

Probabilistic Language Models: Applications

* Can be used to determine the most plausible sentence by assigning a probability to sentences.

- Speech recognition
  - P(I bought fresh mangoes from the market) $\gg$ P(I bot man goes from the man kit)
  - P(I love eating spicy samosas) $\gg$ P(eye love eat tin spy sea same o saus)

- Machine Translation
  - P(Heavy rainfall) >> P(Big rainfall)
  - P(The festival of lights) >> P(The festival of lamps)
  - P(family gatherings) > P(family meetings)
- Context sensitive spelling correction
- Natural Language generation, etc.



Probabilistic Language Models

* Language model – is a probability distribution over a sequence of tokens (characters, words, subwords, etc)

Goal: Calculate the probability of a sentence or sequence consisting of 'n' words

$$P(W) = P(w_1, w_2 \ldots w_n) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1, w_2) \ldots P(w_n|w_1, w_2 \ldots w_{n-1})$$

Related Task: Calculate the probability of the next word conditioned on the preceding words

$$P(w_6|w_1, w_2 \ldots w_5)$$

Probability of a sentence

eg: The monsoon season has begun.

- How to compute the probability of the sentence?

$P(W) = P(\text{"The monsoon season has begun"})$
$= P(\text{The, monsoon, season, has, begun})$

Calculated using principles of 'Chain rule of Probability'

Chain Rule of Probability

Conditional Probability $P(A|B) = P(A,B)/P(B)$

Rewriting: $P(A,B) = P(A|B) \cdot P(B)$

More variables $P(A,B,C,D) = P(A) \cdot P(B|A) \cdot P(C|A,B) \cdot P(D|A,B,C)$

Chain rule in general:

$$P(x_1, x_2, \cdots x_n) = P(x_1) \cdot P(x_2|x_1) \cdot P(x_3|x_1,x_2) \cdots P(x_n|x_1,x_2 \cdots x_{n-1})$$

Probability of a sequence

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

$P(W) = P(\text{"The monsoon season has begun"})$
$= P(\text{The, monsoon, season, has, begun})$
$= P(\text{The}) \cdot P(\text{monsoon}|\text{The}) \cdot P(\text{season}|\text{The, monsoon}) \cdot P(\text{has}|\text{The monsoon season}) \cdot P(\text{begun}|\text{The monsoon season has})$

Q) How to compute these probabilities?

A) $P(\text{season}|\text{The monsoon}) = \frac{\text{count}(\text{The monsoon season})}{\text{count}(\text{The monsoon})}$

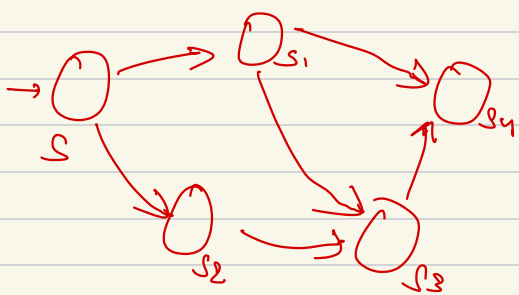
Estimate Conditional Probabilities

$$P(\text{began} \mid \text{The monsoon season has}) = \frac{\text{Count}(\text{The monsoon season has began})}{\text{Count}(\text{The monsoon season has})}$$

Problem: As the context gets longer, it is difficult to find occurences of all the words together. And if one component of the product becomes 0 then the entire probability becomes 0.

Solution: Markov Assumption

First order Markov Assumption:
Transition probability is only dependent on the previous state



k^{th} order Assumption: Every state depends on previous k states

Hence, when we measure the probability: $\frac{\text{Count}(\text{The monsoon season has began})}{\text{Count}(\text{The monsoon season has})}$

we will not look at the entire context but only the previous (or previous 2 or 3) word(s)

Applying this to chain rule:

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

N-gram Language Models

Lets consider the conditional prob: P(begun | the monsoon season has)

An N-gram model considers only the preceding N-1 words

- Unigram: P(begun) [All words are independent. 0th order Markov Assumption]
- Bigram: P(begun | the)
- Trigram: P(begun | the monsoon)

Relation b/w Markov model & Language Model
An N-gram Language Model ≡ (N-1) order Markov Model

NOTE: Unigram assumes that all words are independent but that is not the case in English (or any other language). As the context gets longer, more dependencies are captured but it is highly likely that the probabilities become 0. Hence, there is a trade off & we try to restrict ourselves to 3-4 gram models

Lets focus on the bigram model

* How to measure the probability?

$$P(w_2|w_1) = \frac{P(w_1, w_2)}{P(w_1)}$$

$$= \frac{P(\text{bigram})}{P(\text{unigram})}$$

Raw bigram counts (absolute measure)

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Raw unigram counts (absolute measure)

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

Unigram and bigram counts for eight of the words (out of V = 1446) in the Berkeley Restaurant Project corpus of 9332 sentences. Previously-zero counts are in gray.

↳ square matrix

* 1st word is not counted in the columns & last word is not counted in the rows

* To do this we need a table of bigram & unigram counts as shown above in the table. (constructed from the entire corpus of documents)
* The number circled in red, indicates how many times 'want' occurs after 'i'. (The table is asymmetric i.e., 'i want' is not the same as 'want i')

Q) Can we guess the unigram count by looking at the bigram table?

Q) Is the sum of entries of the row for a word the same as the sum of entries for a column?

A) Yes

(think of the edge cases - sum can only be different when any word (let's say 'I') is the 1st* or last word* in a sentence. But we also have a <start> token for the start of the sentence & <end> token for end of sentence so it gets counted here. (Hence, the sum of row & column for a particular word will always be the same)

Probability table can be calculated by dividing the rows with the respective unigram counts

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

* This matrix is very sparse.
  - But that doesn't necessarily mean all pairs with a count of 0 are invalid.
    - But a lot of them will be syntactically or semantically incorrect. (eg: 2 verbs like 'eat want' cannot appear together)

* 'to' is generally a grammatical follow up to 'want'.

* The above matrix is not just a statistical matrix but highlights a lot of important things.
  - Want to - very high probability (grammatical inference) *
  - Chinese food - very high probability (world knowledge)
  - to food - (contingent zero ⇒ i.e., difficult to think of a sentence where these 2 words appear together. (But not impossible: eg- Goo to food festival.))

Limitations of N-gram models

* Insufficient as they're unable to capture long range dependencies present in the language.

Eg: The project, which he had been working on for months, was finally approved by the committee.

Here 'approved' refers to the 'project' but a trigram or 4-gram language model will fail to predict the next word.

P(approved | months was finally) = highly unlikely

Estimate N-gram Probabilities

* Maximum Likelihood Estimate (MLE)
  - Used to estimate params of a statistical model
  - Determine the most likely values of params that would make the observed data most probable.

eg: bigram probabilities are estimated as follows:

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

Limitations of MLE Estimation

Problem: N-grams work well when test corpus is like train corpus which is often not the case in reality (data sparsity issue)

Train set:	Test set
• ... enjoyed the movie	• ... enjoyed the concert
• ... enjoyed the food	• ... enjoyed the festival
• ... enjoyed the game	• ... enjoyed the walk
• ... enjoyed the vacation	

Zero probability n-grams
- P(concert | enjoyed the) = P(festival | enjoyed the) = P(walk | enjoyed the) = 0
- As a result the probability of test set will be 0.
- Perplexity cannot be computed (cannot divide by 0)

* In fact, it is possible that a word that appears in the test set is not in the vocabulary itself i.e., in the matrix (because matrix & vocabulary are constructed from the training set only)
* Such tokens are called OOV (out-of-vocabulary) tokens

Q) How to deal with such words?

[Idea: Look for similar words in the same context & use it as proxy for the unknown word.]

A) Generally, what we do is we maintain a lexicon.
Lexicon - contains only those words whose frequency is above a certain threshold (from the vocabulary)
All tokens whose frequency is below the threshold is considered as a proxy for the unknown

Unknown tokens = Vocabulary − Lexicon

Bigram table now has entries from the lexicon & an entry for all unknown words denoted by a token <UNK>

Replace all the entries of vocabulary tokens with frequency below the threshold with <UNK> and re-calculate the bigram table.

Q) The matrix is very sparse. How to fix this? (99% entries are 0)
A) Smoothing

Smoothing Techniques

Laplace Smoothing (Add one estimation)

* Imagine that we encountered each word (N-gram) one more time than it's actual occurence
* Simply increase all counts by 1.

MLE Estimate (for bigram)

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

Add 1 estimate

$$P_{Add-1}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + |V|}$$

$|V|$ → length of vocabulary

This is needed to adjust the probabilities so that $\sum P(i|e) = 1$ i.e., sum of all probabilities is still 1. To look at it in another way: if we increment the count of each bigram by 1, then how much has the unigram frequency increased. The answer is $|V|$ i.e., the length of vocabulary.

[In reality we don't actually update the unigram count table but only the effective bigram count]

Effective bigram count ($c^*(w_{n-1}\, w_n)$):

$$\frac{c^*(w_{n-1}\, w_n)}{C(w_{n-1})} = \frac{C(w_{n-1}, w_n) + 1}{C(w_{n-1}) + |V|}$$

In a more general sense, add k estimate

$$P_{Add\text{-}k}(w_i | w_{i-1}) = \frac{C(w_{i-1}, w_i) + k}{C(w_{i-1}) + kV}$$

Compare with bigrams: Before & After Smoothing

	i	want	to	eat	chinese	food	lunch	spend
	5	827	0	9	0	0	0	2
	2	0	608	1	6	6	5	1
	2	0	4	686	2	0	6	211
	0	0	2	0	16	2	42	0
	1	0	0	0	0	82	1	0
	15	0	15	0	1	4	0	0
	2	0	0	0	0	1	0	0
	1	0	1	0	0	0	0	0

Bigram count before

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Effective bigram count

→ Essentially like a steal from rich & give to poor kind of smoothing.

This becomes an issue because the actual counts are not reflected in the matrix anymore i.e., there is a huge difference from the original number

More General Smoothing Techniques

*) Add-k smoothing:

$$P_{Add\text{-}k}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i) + k}{C(w_{i-1}) + k|V|}$$

$$P_{add\text{-}k}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i) + m\left(\frac{1}{|V|}\right)}{C(w_{i-1}) + m}$$

where $k = \frac{m}{|V|}$

$m = k|V|$

$\frac{1}{|V|}$ is a uniform which is multiplied to some constant 'm' & added to count of each word

*) Unigram prior smoothing

$$P_{unigram\ prior}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i) + m\,P(w_i)}{C(w_{i-1}) + m}$$

Essentially the same as above but instead of adding uniform $\frac{1}{|V|}$ we add a unigram probability. So, it is now dependent on the current word (w_i) that we are considering

An optimal value for k or m can be determined using a held out dataset (i.e., hyperparameter tuning on validation dataset)

Back off & Interpolation

*) As N increases, N-gram becomes more powerful but incapable of accurately estimating params due to sparsity problem.

*) Often we have limited knowledge of longer contexts, it is helpful to consider less context.

Back-off:

- Opt for a trigram when there is sufficient evidence, otherwise use bigram, otherwise unigram
  - This may not result in a true probability value (i.e., may not sum up to 1)

Interpolation

Linear interpolation

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

$$\sum_i \lambda_i = 1$$

not a true probability

Linear combinat^n of unigram, bigram, trigram

Context-dependent interpolation

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

lambda's are dependent on the context

Lecture 3.2 — Advanced Smoothing & Evaluation

Advanced Smoothing Algorithms

*) Naive smoothing algorithms are not very effective & not used frequently for N-grams

*) Can be used in domains where no. of zeros is not so huge.

Popular algorithms are → Good Turing
→ Kneser-Ney

Use the count of things we've seen once to help estimate the count of unseen

Notation
N_c = Frequency of frequency of c

eg: Rohan I am Iam I like to play.

I	3
Rohan	2
Am	2
like	1
to	1
play	1

$N_1 = 3$ (count of frequency 1)
$N_2 = 2$ (count of frequency 2)
$N_3 = 1$ (" " " 3)

Good Turing Smoothing Intuition

Q) You are birdwatching & see the following: 10 flamingos, 3 kingfishers, 2 Indian rollers, 1 woodpecker, 1 peacock & 1 crane (18 birds)

Q) How likely is it that the next bird you see is a woodpecker?

A) MLE: P(woodpecker) = $\frac{1}{18}$ (This is MLE. The best possible probability from the observation)

Q) How likely that next bird is new species - Purple Heron or Painted Stork?

A) MLE = $\frac{0}{18} = 0$

However, Good Turing says that to estimate the count of

we will look at count 1.

$N_1 = 3$ $\quad\therefore\quad P(\text{new species}) = \frac{3}{18} = \frac{1}{6}$

* Assuming so, how likely is it that the new species is a woodpecker?
  - ↳ Must be less than 1/18 (Because we already took something from the probability mass of woodpecker)

Good Turing Calculations

* P^*_{GT} (things with zero frequency) $= \frac{N_1}{N}$
* Unseen (Purple Heron or Painted Stork)
  - $C = 0$
  - MLE $p = 0/18 = 0$
  - P^*_{GT} (unseen) $= N_1/N = 3/18$
* for others, the modified count will be

$$C^* = \frac{(C+1)N_{C+1}}{N_C}$$

So, for eg., for $C = 3$: $\quad C^* = \frac{(3+1)N_4}{N_3}$

$C = 2$: $\quad C^* = \frac{(2+1)N_3}{N_2}$

*) P(woodpecker) which is seen once

C = 1

MLE p = 1/18

c^*(woodpecker) = $\frac{(1+1)N_2}{N_1} = 2 \times \frac{1}{3} = \frac{2}{3}$

modified count

P^*_{GT}(woodpecker) = $\frac{N_1}{N} = \frac{2/3}{18} = \frac{1}{27}$

modified probability (will follow all the probability rules)

Good Turing Estimation

*) Numbers from Church & Gale (1991)

*) 22 million words of AP Newswire

NOTICE how different it is from Add-1 smoothing. The counts don't change drastically from the original value

Count c	Good Turing c*
0	.0000270
1	0.446
2	1.26
3	2.24
4	3.24
5	4.22
6	5.19
7	6.21
8	7.24
9	8.25

If we observe carefully,

$c^* = c - 0.75$ (roughly)

If this is the case then why do we need all the math above? Why not just subtract a constant (like 0.75) from the original counts?

Absolute Discounting Interpolation

*) Adjusts the probability estimates for n-grams by discounting each count by a fixed amount (usually a

* λ is a hyper parameter

small constant) before computing probabilities.

unigram probability

$$P_{\text{AbsoluteDiscounting}}(w_i \mid w_{i-1}) = \frac{c(w_{i-1}, w_i) - d}{c(w_{i-1})} + \lambda(w_{i-1})\, P(w_i)$$

(like 0.75)

↳ As explained in interpolation before, it is a combinat^n of bigram & unigram

*) But this is not easy. (eg: unigram counts for stop words like 'a', 'the' will be very high)

Continuation Probability

*) Intuition: Shannon game
- My breakfast is incomplete without a cup of ...: coffee/Angeles?
- Says in the corpus 'Angeles' is more prevalent than "coffee"
- However, it is important to note that "Angeles" mostly comes after "Los"

*) Instead of regular unigram probability use <u>continuat^n probability.</u>

↳ measure the no. of unique bigrams that the unigram (under considerat^n) completes.

- Regular unigram probability: $P(w)$: "How likely is w?"
- $P_{\text{continuate}}(w)$: "How likely is w to appear as a novel continuat^n?"

i.e., look at all possible w_x which is completed by the word w $(w_1 w, w_2 w, w_3 w \ldots w_n w)$

↳ so the novel continuat^n count is 'm'.

* How to compute continuation probability?

- Count how many different bigram types each word completes
  ⇒ Normalize by the number of word bigram types

$$P_{continuation}(w) = \frac{|\{w_{i-1} : c(w_{i-1}, w) > 0\}|}{|\{(w_{j-1}, w_j) : c(w_{j-1}, w_j) > 0\}|}$$

No of unique bigrams

All the words such that Count of (w_{i-1}, w_i) is greater than 0

A common word (Angeles) appearing in only one context is likely to have low continuation probability

Kneser-Ney Smoothing

Similar to absolute discounting above but we take max (discounted value, 0) because we don't want -ve numbers

Replace unigram with continuation probability

$$P_{KN}(w_i | w_{i-1}) = \max\left(\frac{c(w_{i-1}, w_i) - d, 0}{c(w_{i-1})}\right) + \lambda(w_{i-1}) P_{continuation}(w_i)$$

where λ is a normalizing constant (How to determine this?)

→ λ is a funct^n of w_{i-1}

→ $P_{KN}(w_i | w_{i-1})$ is a probabsility

→ We have already discounted d from the 1st component

→ So, how many discounts have happened for one w_{i-1}
(Only the no. of places with non-zero bigram counts whose value is greater than 'd')

'd' is a very small value so this is sometimes taken as 0.

Hence, $\lambda(w_{i-1}) = \frac{d}{c(w_{i-1})} \cdot |\{w : c(w_{i-1}, w) > d\}|$

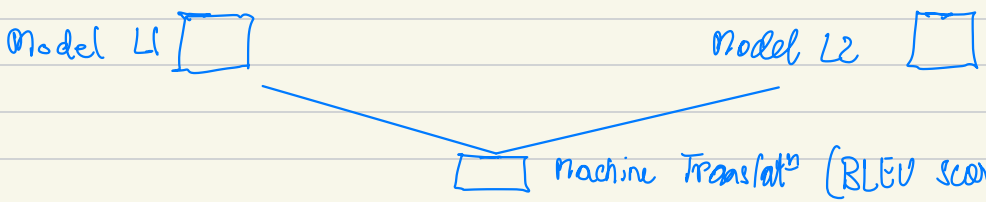
↳ sum of values of the bigram row w_{i-1}

↳ Guarantees that the probability will sum up to 1.

Evaluation of language Models

* 2 types
  - ↳ Extrinsic Evaluation
  - ↳ Intrinsic Evaluation

Extrinsic Evaluat^n: (Task based)



↓

Feed the output of both models to the BLEU score & see which one performs better.

What is the problem?

- ↳ If the quality of translat^n model is poor, we won't be able to assess the actual performance of L1 & L2.
- ↳ There are hundreds of tasks – which one should we choose? (Generally, we consider 10-12 different tasks & see if a model outperforms another in majority of the tasks)
- ↳ Tasks are time taking (Evaluat^n can take a long time)

Intrinsic Evaluation : Perplexity

Intuition : The Shannon Game

How well can we predict the next word?
- I always order pizza with cheese and . . .
- The president of India is . . .
- I wrote a . . .

We see that for the 1st sentence the set of words will be much more limited than for the last sentence
Observation : The more context we consider, the better the predictions

> A better model is characterised by its ability to assign a higher probability to the correct word in the given context.

Perplexity

* The best model is one that best predicts an unseen test set.

Say, Language model L_1 & L_2 are trained on training set (T)

Lets say we have a test set with 'n' sequence of sentences
$W_1\ W_2\ \ldots\ W_n$

We measure the probability of the sequence of 'tokens' based on L_1 & L_2 seperately.

Perplexity $PP(W_{1:n}) = \frac{1}{\sqrt[n]{P(w_1 w_2 \cdots w_n)}} = P(w_1 w_2 \cdots w_n)^{-1/n}$

If probability is high, perplexity will be low and vice versa.

For bigram language model,
Perplexity $= \left(P(w_1) P(w_2|w_1) \cdots P(w_n|w_{n-1})\right)^{-1/n}$

> Perplexity: inverse probability of the test data, normalized by the number of words

Why suddenly the need for perplexity?

What is the n-th root of probabilities: Geometric mean

Perplexity: Inverse geometric mean.

Perplexity & Entropy

Lets say an event X has 'n' outcomes with some probability.

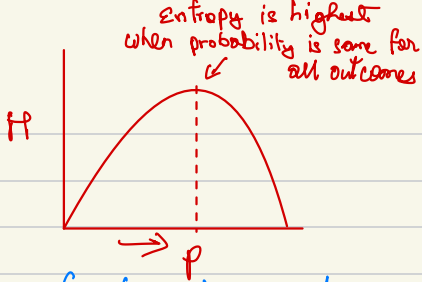
Shannon Entropy $H(X) = -\sum_{n} p(x_i) \log(p(x_i))$

$x_i \in X$

Entropy: measure of randomness

- So, for a uniform vs biased outcome which will have higher entropy?

Naturally, the uniform outcome will have higher entropy (because outcome is more random)



For a language, i.e., a sequence of tokens (sentence) what is the entropy?

$$H(w_1 w_2 \cdots w_n) = -\sum p(w_1 w_2 \cdots w_n) \log p(w_1 w_2 \cdots w_n)$$

sequence of words is the random variable

$w_{1:n} \in L$ → language

But this is entropy of sequence. We need entropy of word

$$\text{Entropy rate / Per word entropy} = \frac{1}{n} H(w_1 w_2 \cdots w_n)$$

$$= -\frac{1}{n} \sum p(\ldots) \log p(\ldots)$$

But here we consider only those cases where sequence length is 'n' but a document may have sequences of multiple sizes. So, how to generalise?

$$-\frac{1}{n} H(w_1 w_2 \cdots w_n) = -\frac{1}{n} \sum p(\ldots) \log p(\ldots)$$

$$= -\lim_{n \to \infty} \frac{1}{n} \sum p(\ldots) \log p(\ldots)$$

Language is a stochastic process which generates tokens based on a distribution we don't know (like we don't know $p()$ here)

$$H(L) = -\lim_{n\to\infty} \frac{1}{n} \sum p(\ldots) \log p(\ldots)$$

Acc. to Shannon-McMillan-Breiman Theorem, if the stochastic process is regular (stationary & ergodic (google for more info)), we can approximate the above value with a sentence that is significantly large. So, we don't need to sum over all sentences. We can approximate it by a sentence whose length is significantly large.

$$\therefore H(L) = -\lim_{n\to\infty} \frac{1}{n} p(\ldots) \log p(\ldots)$$

We also assume, $p(w_1, w_2 \cdots w_n) = \frac{1}{n}$

$$H(L) = -\lim_{n\to\infty} \frac{1}{n} \log p(w_1, w_2 \cdots w_n)$$

Now that we have entropy, Lets define cross entropy.
If there are 2 random variables that define 2 events, cross entropy measures the similarity/closeness b/w the 2 events.

In language, we don't know the actual (or gold standard) probability of generating a word. But we approximate this process using a language model.

$$\therefore H(L, M) = -\sum p_L(x_i) \log p_m(x_i)$$

L: language (actual), M: model

Again taking per-word entropy & applying Shannon-McMillan-Breiman Theorem

$$= -\frac{1}{n} \sum P_L(x_i) \log p_m(x_i)$$

$$\lim_{n \to \infty}$$

$$H(L, M) = \lim_{n \to \infty} -\frac{1}{n} \log p_m(w_1 w_2 \cdots w_n)$$

Cross entropy has an interesting property. It has a bound (i.e., a lower bound)

$$H(L) \leq H(L, M)$$

↳ How?

$H(L)$ is the true entropy of the language that we do not know

$H(L, M)$ is the cross entropy b/w language & model

KL Divergence: Diff b/w L & M

So, if we reduce the diff b/w true language (L) & language model (M), model will come closer to the language

$$H(L) \leq H(L, M)$$

↳ model

↓ True probability

$$H(L) = -\sum p_i \log p_i$$

$$= -\sum p_i \log \frac{p_i}{m_i} \cdot m_i$$

$$= -\sum p_i \log m_i - \sum p_i \log \frac{p_i}{m_i}$$

cross entropy ↵

↳ KL Divergence b/w p & m

$$\therefore H(L) = H(L, M) - KL(L, M)$$

Hence, $H(L) \leq H(L, M)$

Lets focus on the original eqn

$$H(L,M) = \lim_{n\to\infty} -\frac{1}{n} \log p_M(w_1 w_2 \ldots w_n)$$

To simplify, we ignore the limit

$$H(L,M) = -\frac{1}{n} \log p_M(w_1 w_2 \ldots w_n) \quad -(1)$$

We know that Perplexity (PP) $= p(w_1 w_2 \ldots w_n)^{-1/n}$ $-(2)$

And from (1), we can say

$$2^H = 2^{-\frac{1}{n}\log p(w_1 w_2 \ldots w_n)}$$
$$= 2^{\log p(w_1 w_2 \ldots w_n)^{-1/n}}$$
$$= p(w_1 w_2 \ldots w_n)^{-1/n} \quad (\log_2 \text{ and } 2^x \text{ cancelled})$$
$$= PP$$

$$\therefore PP = 2^H$$

Intuitively, think of it this way:

Entropy = Avg no. of bits required to encode an information

$\therefore -\log p_i$ or $\log \frac{1}{p_i}$ = no of bits required to encode

Entropy $= -p_i \log p_i$ = Expected no of bits required
(Expectation $= \sum x_i p(x_i)$)

$2^{\text{expected no of bits}}$ = no of possible outcomes = PP

↳ (no of possible words that can be generated with those bits)

∴ Perplexity ≈ Possible no of words we can generate

So, lets say we have a set of tokens & we want to estimate the next word.

If entropy is high, perplexity is also high & vice versa

Q) So, why do we need perplexity? why not directly use entropy?
But entropy only measures the avg. no of bits required to encode the output. Whereas, perplexity tells us the actual no. of possible values/tokens that can be generated for the next word. So, it makes more sense to use as an evaluation metric.

Problems of Statistical Language Models

*) N-gram LMs suffer from data sparsity & limited context.
  - → Predicting the next word from a fixed window of previous words
  - → Fixed context size: Limited to a fixed window of previous words

*) As the no of bigrams increases, matrix becomes huge.
*) Out-of-Vocabulary (OOV) word handling is very bad.
*) Computationally very expensive for larger n-gram models.

Need for richer representations

* Contextual Understanding: Need for models that understand context beyond fixed windows
* Semantic similarity: Ability to capture relationship b/w words (eg: synonyms)
* Scalability: Models that can scale to large datasets & vast vocabularies efficiently.

Word Embeddings - Shows how representing words (tokens) as vectors and transition to neural LMs solves many of these problems

---

Lecture 4.1 - Word Representation : Word2Vec

'Meaning' of a word
↳ The idea represented by the word.

Need for word representation
* For language modeling:
  ↳ We need effective representation of words
    ↳ The representation must somehow encapsulate the word meaning

Representing words as discrete symbols
* Traditional NLP regards words as discrete symbols
  ↳ Ontologies were created to represent the relationship b/w words (like hologram, meronym, antonym, synonym, etc.)
  ↳ Whenever we're interested in the meaning of a word, we retrieve the ontology, extract neighboring words & try to guess its meaning

* But the above method is not the most ideal & what we need is a kind of vectorized representation of the word.

* Can we just take the concatenated ASCII values as the representation?

* But then the length of representation for diff size words would be different.

* Simplest method - One hot encoding

↳ We have a dictionary in which all the words are listed with an id.
Lets say the size of dictionary is 1 million.
The idea is that we create a vector of size 1 million for each word & the all the numbers in that vector will be zero except one position, i.e. the id of that word.

∴ motel = [0 0 0 0 0 0 1 0 0 0 0 0]
hotel = [0 0 0 0 1 0 0 0 0 0 0 0]

- Hence, each word has a unique representation

Issues

↳ All vectors are orthogonal i.e., dot product will be 0 (just like in above vector). But the words like hotel & motel which are semantically similar should have similar representations.
Also, the vector size will increase if the dictionary size increases & we may need to recompute the vectors again.

Solutn: Can we make use of WordNet's list of synonyms to get similarity?

Using existing Thesauri or Ontologies like Wordnet

WordNet 3.0

* A hierarchically organised lexical database.
* Online thesaurus + aspects of a dictionary
  ↳ some other languages also available (Arabic, finnish, etc.)

Category	Unique Strings
Noun	117,798
Verb	11,529
Adjective	22,479
Adverb	4,481

How is 'sense' defined in WordNet?

↳ Each sense (or meaning) of a word is represented by a synset
↳ The descriptn of the synset is called gloss
↳ eg:- chump as a 'noun' with the gloss:
"a person who is gullible & easy to take advantage of"
↳ descriptn of 1 sense of the word chump

- This sense of "chump" is shared by 9 words:
chump, fool, gull, mark, patsy, fall guy, sucker, soft touch, mug
- Each of these senses have this same gloss.
(But not every sense; eg: another sense of gull is aquatic bird)

Issues with Thesaurus based Approaches

* Other meanings might not be captured in diff. versions.
* Missing nuance.
eg: 'good' is listed as a synonym for 'proficient' but this is true only in some contexts.

Example:

Senses of 'bass':

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations
Display options for sense: (gloss) "an example sentence"

Noun

- S: (n) **bass** (the lowest part of the musical range)
- S: (n) **bass**, bass part (the lowest part in polyphonic music)
- S: (n) **bass**, basso (an adult male singer with the lowest voice)
- S: (n) sea bass, **bass** (the lean flesh of a saltwater fish of the family Serranidae)
- S: (n) freshwater bass, **bass** (any of various North American freshwater fish with lean flesh (especially of the genus Micropterus))
- S: (n) **bass**, bass voice, basso (the lowest adult male singing voice)
- S: (n) **bass** (the member with the lowest range of a family of musical instruments)
- S: (n) **bass** (nontechnical name for any of numerous edible marine and freshwater spiny-finned fishes)

Adjective

- S: (adj) **bass**, deep (having or denoting a low vocal or instrumental range) *"a deep voice"; "a bass voice is lower than a baritone voice"; "a bass clarinet"*

Wordnet also lists some offensive synonyms in some synsets w/o any coverage of connotations or appropriateness of words

* Missing new meanings of words
  ↳ Impossible to update because of the required manual efforts.
* Subjective (depends on the annotator)
* Requires manual labor
* Also doesn't cover other parts of speech beyond noun, verb, adverb, adjective properly.

Representing Words by Their context

Distributional Semantics: A words meaning is given by the words that frequently appear close-by.

↳ The premise on which most of the existing word representation techniques are proposed.

"You shall know a word by the company he keeps" - JR Firth (1957)

* When a word w appears in a text, its context is the set of words that appear nearby (within a fixed size window)
  ↳ We can have many contexts of w to build up a representation of w.
    ↳ ...govt debt problems turning into banking crises as happened in 2009...
    ↳ ...saying that Europe needs unified banking regulation to replace the hodgepodge...
    ↳ ...India has just given its banking system a shot in the arm...
  ↳ These context words will represent banking

Count based Methods

Use co-occurences for Word Similarity

The Term-Context matrix (or, word-coord matrix)

* Each cell: no of times the row (target) word and the column (context) word co-occur in some contexts in the corpus
  - Generally, smaller contexts are used, like:
    - paragraph
    - Window of 10 words

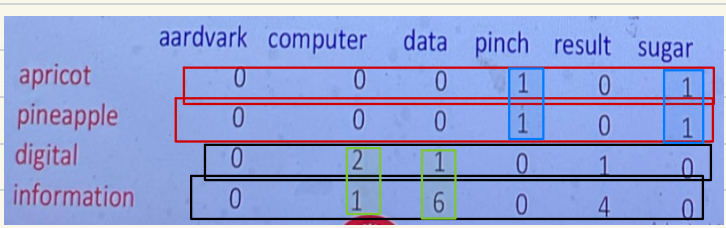
N^V	aardvark	computer	data	pinch	result	sugar
apricot	0	0	0	1	0	1
pineapple	0	0	0	1	0	1
digital	0	2	1	0	1	0
information	0	1	6	0	4	0

* Each word is a count vector in N^V: a row below (V: size of vocab)

(Similar to bigram count, except that in bigram count words should appear side-by-side)

- Row vector can be treated as the term vector
- Column " " " " " " context "
- The choice of which vectors to take is yours
  - Can take both vectors & sum them.
  - Try with both & see which gives better results

* If we look at 2 entries, like apricot & pineapple, they both appear with the word pinch & sugar but not with words like computer & data. So, the words might be similar
* Similarly, the words digital & information co-appear with computer & data but not with aardvark, pinch & sugar.



* We can take a pair of rows/columns, do some similarity measure (like cosine similarity) & identify how similar the words are.

Issues

* Again matrix size increases with increasing no of terms
* Vector will be large if there are many words
* Matrix is very sparse
* Stop words (like a, an, the) will have very high corresponding entries.

* We'd rather have a measure that asks whether a context word is particularly informative about the target word.

* For term-document matrix, we generally use tf-idf instead of raw term counts.

TF-IDF

Term frequency

$tf_{t,d} = count(t,d)$

t → term
d → document

count of term 't' in document 'd'
↳ document can be a page, a book, a paragraph, etc.

Instead of using raw count we squash it a bit:

$tf = \log_{10}(count(t,d)+1)$

↳ for smoothing

Issue with tf: frequently occuring words (like stop words) will have high counts.

Document frequency

df_t is the number of documents t occurs in.

(NOTE: This is not collection frequency: total count across all documents)

eg: "Romeo" is very distinctive for one Shakespeare play:

	Collection frequency	Document frequency
Romeo	113	1
action	113	31

→ frequency sum across all documents

Inverse document frequency

*) We take the inverse because if a word appears in more documents, it is most likely a common word (like STOP words) & thus, it's importance will be reduced.

*) We should choose words that are representative

$$idf_t = \log_{10}\left(\frac{N}{df_t}\right)$$

N => no of documents

Can sometimes also be $\log_{10}\frac{N}{df_t+1}$ or $\log_{10}\left(\frac{N}{df_t}+1\right)$ for smoothing & to avoid errors.

final tf-idf weighted value

$$w_{t,d} = tf_{t,d} \times idf_t$$

Raw counts	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

tf-idf	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

* So, words like Julius Caesar which have low count, still has high tf-idf but common words like fool despite the high count have low tf-idf score.

* Each row here is a vector.

* Addresses the issue of frequent words but there are still other problems.

Issues → Needs recomputation & vector (and matrix) size will increase with new words & documents.
→ Sparsity

Drawbacks of Co-occurence Matrix (or count based) Approach

* Quadratic Space needed
* Relative position & order of words not considered

Low dimensional vectors

* Store only important information in fixed, low dimensional vector.
* SVD on co-occurence matrix.

Drawbacks → Computation cost still scales quadratically for $n \times m$ matrix: $O(mn^2)$ flops (when $n < m$)
→ Hard to incorporate new words or documents (recomputation cost)
→ Does not consider order of words

Prediction-based Methods

Word Embedding → Idea come around 2013

* Dense vector
* Helps in learning less parameters (less prone to overfitting)
* May generalize better
* Can capture synonyms better
  - car & automobile are synonyms; but have distinct dimensions
  - A word with car as neighbor & a word with automobile as neighbor should be similar, but are not.

Represent the meaning of word : Word2Vec

* Word2Vec was proposed in 2013.
* 2 approaches to Word2Vec:
  - Continuous Bag of Words (CBOW): use a window of words to predict the middle word
  - Skip-gram (SG): use a word to predict the surrounding words of window

CBOW

use

eg: I am going to school

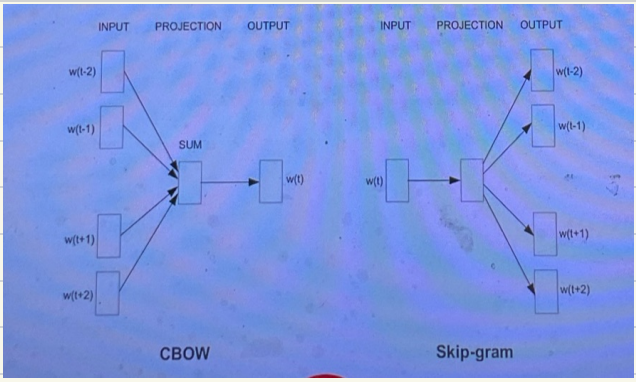
predict

SKIP-GRAM

predict

eg: I am going to school

use



The more common approach we use is skip-gram with negative sampling

* Instead of counting how often each word 'w' occurs near 'apricot'
  - Train a classifier on a binary prediction task:
    - Is 'w' likely to show up near 'apricot'?

* We don't care about the task itself but we will use the generated classifier weights as the word embeddings.

* Big idea: Self-supervision
  - A word 'c' that occurs near apricot in the corpus acts as the gold "correct answer" for supervised learning
  - No need for human labels
  - Bengio et. al (2003); Collobert et. al (2011)

Approach: Predict if candidate word 'c' is a "neighbor"

1. Treat the target word 't' and a neighboring context word 'c' as positive examples (within a target window of fixed size, say 7)

The middle word is the target & the remaining 6 words around it are the context

$$\cdots w_{t-4} \boxed{w_{t-3}\ w_{t-2}\ w_{t-1}\ w_t\ w_t\ w_{t+1}\ w_{t+2}\ w_{t+3}} w_{t+4} \cdots$$

Positive pairs: (w_t, w_{t-3}), (w_t, w_{t-2}), (w_t, w_{t-1}), (w_t, w_{t+1}), (w_t, w_{t+2}), (w_t, w_{t+3})

2- The above 6 are context words & all the remaining N−7 words in the vocabulary (N → size of vocabulary) are non-context words
Randomly sample other words in the lexicon to get negative examples

↳ Specifically in skip-gram we sample 'k' times more negative samples (as it gives more evidence; also sometimes we may sample -ve examples, some +ve samples & infact the same word can also come as context. So, it's always safe to consider more words as context.)

3- Use logistic regression to train a classifier to distinguish b/w the 2 cases (+ve & -ve)

Keep moving the window - create the positive pairs, sample the -ve pairs, run the classifier and move forward until the entire document is scanned.

4. Use the learned weights as embeddings.

Skip-Gram Training Data

*) Assume a +/- 2 word window, given training sentence:

... lemon, a [tablespoon of apricot jam, a] pinch
c_1 c_2 target c_3 c_4

*) For every word 'w' we have a context vector C
Remember, each word here is represented by a vector

which we are going to learn through the classificatn task.

*) We need a representatn of 'w' & a representatn for 'c' ($c \in C$ i.e., each word vector in the context) which we learn
*) We can start with a random initializatn or a one hot vector

Goal: Train a classifier, that given a candidate pair (word, context)
(apricot, jam)
(apricot, aardvark)
...

assigns each pair a probability:

$P(+|w,c)$

$P(-|w,c) = 1 - P(+|w,c)$

Similarity Computation

*) Similarity is computed using dot product
Remember: 2 vectors are similar if dot product is high
Cosine is just a normalized dot product

*) Similarity $(w,c) \propto (w \cdot c)$
*) We need to normalize to get probability
↳ Cosine isn't a probability either

Turning dot product into probabilities

*) Similarity $(w,c) \approx w \cdot c$
*) To turn this into a probability:
↳ we use the sigmoid functn as in logistic regression:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

* Because $1 - \sigma(z) = \sigma(-z)$

$$P(+|w, c) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)}$$

$$P(-|w, c) = 1 - P(+|w, c)$$
$$= \sigma(-c \cdot w)^{*} = \frac{1}{1 + \exp(c \cdot w)}$$

Sigmoid is used because it squashes everything b/w 0 & 1 and the derivative has some nice properties

$$\frac{d}{dz}\sigma(z) = \sigma(z)(1 - \sigma(z))$$

* But this is just for 1 pair. There are multiple such pairs (both +ve & -ve)

Q) So, how do we calculate $P(+|w, c)$?

A) We assume all the pairs are independent & just multiply them.

$$P(+|w, c_{1:L}) = \prod_{i=1}^{L} \sigma(c_i \cdot w)$$

$$\log P(+|c_{1:L}) = \sum_{i=1}^{L} \log \sigma(c_i \cdot w)$$

Skip-gram Classifier: Summary

* A probabilistic classifier, given
  - a test target word w
  - its context window of L words $c_{1:L}$

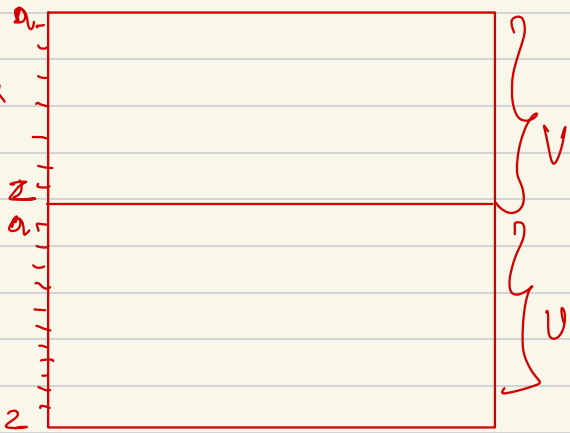
*) Estimates probability that w occurs in this window based on similarity of w to $c_{1:L}$ embeddings

*) To compute this, we just need embeddings for all the words

Q) How many vectors do we compute for each word?
A) 2 vectors - ① As a target word 'w'
② As a context word 'c'

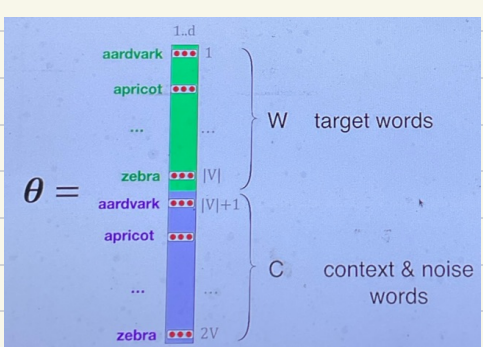
These embeddings are the parameters that we learn

Thus, if we think of the embeddings as a dense matrix with vocabulary size as V, then we are learning a total of 2V vectors i.e., V for target words 'w' & V for context words 'c'.



The size of the vector is a hyperparameter.

The Embeddings we Need: A set for 'w' & a set for 'c'



*) If we say that the word apricot is a positive context for the target word aardvark, it means that as we train, we try to move the context embedding of apricot closer to the target embedding of aardvark.

*) Also, the negative example context embeddings should move farther away from the target word.

Word2Vec: Learning the Embeddings

*) Assume a +/- 2 word window, given training sentence.

... lemon, a [tablespoon of apricot jam, a] pinch ...

c_1 c_2 target c_3 c_4

+ve samples

t	c
apricot	tablespoon
apricot	of
"	jam
"	a

-ve samples ('k' -ve samples sampled from document)

t	c
apricot	aardvark
"	my
"	where
"	co-axial

t	c
apricot	seven
"	forever
"	dear
"	if

Choosing Negative Examples

*) Sampled based on unigram distribution

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$$

*) Just taking the count would mean that non-contextual words (with high frequency) like stop words will be most likely to be sampled but we do not want that

*) Setting $\alpha = 0.75$ gives better performance because it gives more noise words slightly higher probability for rare words: $P_\alpha(w) > P(w)$

* Here we take the product & maximize both because we are taking the probability of the instance being +ve if the word is +ve & the instance being -ve if the word is -ve.

eg $P(a) = 0.99$ $\quad P_\alpha(a) = \frac{0.99^{75}}{0.99^{75}+0.01^{75}} = 0.97$

$P(b) = 0.01$ $\quad P_\alpha(b) = \frac{0.01^{75}}{0.99^{75}+0.01^{75}} = 0.03$

Word2Vec: How to learn Word Vectors

* Start with +ve & -ve examples & initial embedding vectors
* Goal: Adjust the vectors such that we:

Maximize the similarity of target, context pair (w, C_{pos}) for +ve example

Minimize the similarity of target, context pair (w, C_{neg}) for -ve examples.

Loss fnctn for One w with $C_{pos}, C_{neg1}, C_{neg2} \cdots C_{negk}$

* To maximize similarity of target with actual context words & minimize it with k negative sampled words:

$$L_{CE} = -\log\left[P(+|w, C_{pos}) \prod P(-|w, C_{neg})\right]^*$$

$$= -\left[\log P(+|w, C_{pos}) + \sum_{i=1}^{k} \log P(-|w, C_{neg})\right]$$

$$= -\left[\log P(+|w, C_{pos}) + \sum_{i=1}^{k} \log\left(1 - P(+|w, C_{neg})\right)\right]$$

$$= -\left[\log \sigma(C_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-C_{neg} \cdot w)\right]$$

↳ from Pg 60

Minimize the -ve log likelihood ↩

Parameters are w & C.

Gradient Descent

* At each step,
  - Direction: we move in the reverse direct^{n} from the gradient of the loss funct^{n}
  - Magnitude: we move the value of this gradient $\frac{d}{dw} L(f(x;w),y)$ weighted by a learning rate η
  - Higher learning rate means w moves faster.

$$w^{t+1} = w^{t} - \eta \frac{d}{dw} L(f(x;w),y)$$

Derivative of the loss funct^{n}

$$L_{CE} = -\left[\log \sigma(c_{pos} \cdot w) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i} \cdot w)\right]$$

$$\frac{\partial L_{CE}}{\partial c_{pos}} = \left[\sigma(c_{pos} \cdot w) - 1\right] w \quad - (1)^{*}$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = \left[\sigma(c_{neg} \cdot w)\right] w \quad - (2)$$

$$\frac{\partial L_{CE}}{\partial w} = \left[\sigma(c_{pos} \cdot w) - 1\right] c_{pos} + \sum_{i=1}^{k} \sigma(c_{neg_i} \cdot w)\, c_{neg_i} \quad - (3)$$

* How $-(1)$?

$$\frac{\partial}{\partial a}\left(-\log(\text{sig}(a))\right) \frac{\partial \text{sig}(a)}{\partial a} \frac{\partial a}{\partial c_{pos}} = -\frac{1}{\text{sig}(a)}\left(\text{sig}(a)(1-\text{sig}(a))\right) w = -(1-\text{sig}(a))\, w$$

$$= (\text{sig}(a) - 1)\, w$$

$$= \left[\sigma(c_{pos} \cdot w) - 1\right] w$$

Update Equation in SGD

Start with randomly initialized C & W matrices, then do incremental updates

$$C_{pos}^{t+1} = C_{pos}^{t} - \eta \left[\sigma(C_{pos}^{t} \cdot w^{t}) - 1\right] w^{t}$$

$$C_{neg}^{t+1} = C_{neg}^{t} - \eta \left[\sigma(C_{neg}^{t} \cdot w^{t})\right] w^{t}$$

$$w^{t+1} = w^{t} - \eta \left[\left[\sigma(C_{pos}^{t} \cdot w) - 1\right] C^{t} + \sum_{i=1}^{k} \left[\sigma(C_{neg_i}^{t} \cdot w^{t})\right] C_{neg_i}^{t}\right]$$

2 sets of embeddings

Skip gram learns 2 sets of embeddings:

1. Target embedding matrix W
2. Context embedding matrix C

It is common to just add them together, representing the i^{th} word as the vector $W[i] + C[i]$

NOTE: Word2Vec is a static embedding
There are no different vectors for different senses of the word.
Modern approaches like BERT provide contextual embeddings

Summary

*) Start with a random $2V \times d$ ($d \rightarrow$ dimension of embedding) matrix as initial embeddings
*) Train a classifier based on similarity of embeddings
  - Move sliding window
  - Take +ve & -ve instances
  - Run the logistic regression
  - Update weights
  - Move forward with the window
*) finally, throw away the classifier & keep the embeddings

Diff b/w count based approaches & predicted embeddings

*) Count based approaches, scan the document only once & keep the count. They preserve the statistics very carefully. Word2Vec does no such thing. Matrix is constantly updated as we train.

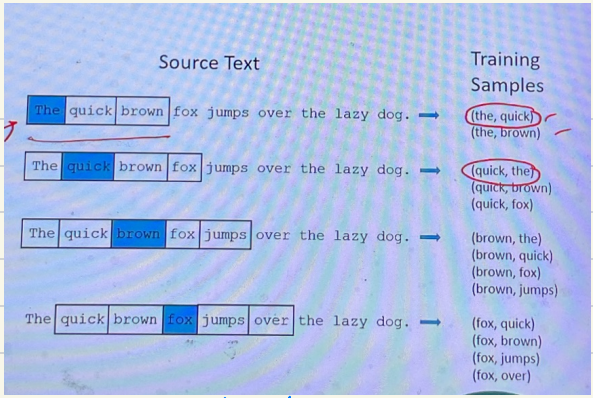
*) As no. of words increase, count based methods need to re compute the whole matrix again.
Word2Vec, we might just have to run the logistic regression one more time & update the embeddings (or add new words)

Some Tricks

Sub-sampling frequent words

There are 2 problems with common words like "the":

1) When looking at word pairs, (fox, the) doesn't tell us much about the meaning of "fox". "the" appears in the context of pretty much everything



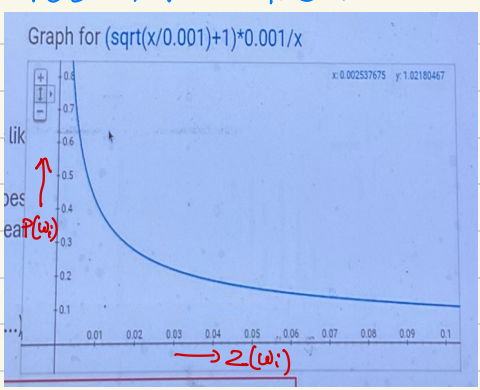
2) We will have many more samples of "the" ((the, quick), (quick, the), (the, lazy), (lazy, the)) than is needed to learn a good vector for "the".

In subsampling,

$P(w_i)$ is the probability of keeping a word:

$$P(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1\right) \cdot \frac{0.001}{z(w_i)}$$

$z(w_i)$ → raw frequency of w_i



Graph for this funct^n

If a word appears more, it is less likely to be sampled

- If we have a window size of 10, and we remove a specific instance of "the" from our text:
  - As we train on the remaining words, "the" will not appear in any of their context windows.
  - We'll have 10 fewer training samples where "the" is the input word.

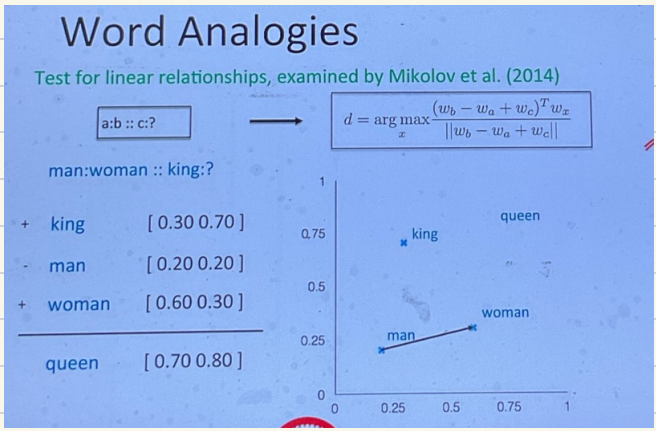
Here are some interesting points in this function (again this is using the default sample value of 0.001).

- $P(w_i) = 1.0$ (100% chance of being kept) when $z(w_i) <= 0.0026$.
  - This means that only words which represent less than 0.26% of the total words will be subsampled.
- $P(w_i) = 0.5$ (50% chance of being kept) when $z(w_i) = 0.00746$.
- $P(w_i) = 0.033$ (3.3% chance of being kept) when $z(w_i) = 1.0$.
  - That is, if the corpus consisted entirely of word w_i, which of course is absurd

Some Interesting Results

*) All sorts of word analogy tests & operations on the embeddings.

eg: emb(king) − emb(man) + emb(woman) ≈ emb(queen)



↳ Beautiful linear relationship emerged

Problems of Word2Vec

*) Static embeddings : i.e.; don't capture context

eg: The cat sat on the mat
Word2Vec can't capture if:
"The" is a special context of words "cat" & "mat"
or
"The" is just a stopword

*) Can't handle unknown words in test dataset. To mitigate this:

fasttext embedding – Subword embedding

- Each word is represented by itself plus a bag of constituent *n*-grams, with special boundary symbols '<' and '>' added to each word.
- For example, with n = 3 the word ***where*** would be represented by the sequence plus the character n-grams:

  where, <wh, whe, her, ere, re> → angular brackets are also considered as characters

- Skip-gram is learned for each constituent *n*-gram eg: nifty: <nif + ift + fty + ty>

where is represented by the sum of all of the embeddings of its constituent n-grams.
Unknown words can then be presented only by the sum of the constituent n-grams

Lecture 4-2 — Word Representations: GloVe

Count based vs Prediction based methods

(not good for embedding)

Count based

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

Prediction based

- Scales with corpus size
- Inefficient use of statistics
- Improved performance on other tasks
- Can capture complex patterns beyond word similarity.

GloVe — Global Vectors

- Try to use the best of both worlds.

Insight: Ratio of co-occurrence probabilities can encode word meaning

	x = solid	x = gas	x = water	x = random
P(x | ice)	large	small	large	small
P(x | steam)	small	large	large	small
P(x | ice) / P(x | steam)	large	small	~1	~1

Jeffrey Pennington, Richard Socher, Christopher D. Manning, "GloVe: Global Vectors for Word Representation", 2014

Idea: We will build the co-occurrence matrix (like in count-based approaches) i.e., when 2 words appear together within a context window, we increase the count.

- But here we do not use the raw counts.

Let's say we are trying to understand some concepts related to thermodynamics and associated words as shown in the figure.

Lets take the words ice & steam.

Q) what other words appear with ice & steam?

A) We get the counts from co-occurrence matrix (All the words that appeared with these words in the context)

Lets say we're looking at 4 context words i.e., solid, water, gas, fashion

	x = solid	x = gas	x = water	x = random
P(x | ice)	large	small	large	small
P(x | steam)	small	large	large	small
P(x | ice) / P(x | steam)	large	small	~1	~1

If ice is the target, it is highly likely that the word solid appears with it. Hence P(x|ice) will be large/high. Similarly, P(x|ice) will be low if x = gas
P(x|ice) will be high for x = water & low for x = fashion.

Similarly, we can't calculate the same for target word steam. These context words are called pivot words & the no of such words = size of vocabulary.

Now, we take the ratios of P(x|ice) : P(x|steam). So, it produces the values as shown in the above fig.
Using these numbers, we can differentiate the relevant (like solid & gas) context words from non-relevant (like water & fashion) context words.

	x = solid	x = gas	x = water	x = random
P(x | ice)	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
P(x | steam)	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
P(x | ice) / P(x | steam)	8.9	8.5×10^{-2}	1.36	0.96

If the numbers tend to 1, then they're non-relevant to differentiate b/w ice & steam (fashion is irrelevant bcoz it occurs with neither & water, bcoz it occurs with both). It also differentiates which context word is important for ice & which for steam (solid is imp for ice so the ratio is large & gas is imp for steam so ratio is low)

* This is asymmetric i.e., $P(j|i) \neq P(i|j)$

The above fig. shows the actual numbers. This concept will be used to derive the GloVe embeddings.

Co-occurence Matrix

Lets denote it as X

(context) ——— j ———→

count (target) ↑ i	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Compute $P(j|i)$ from X, for 2 words i & j in corpus

$$P(j|i) = \frac{X_{ij}}{\sum_j X_{ij}} = \frac{X_{ij}}{X_i}$$

Learn word vectors based on these counts

*) for the 2 words, i & j, assume their corresponding representation vectors are w_i & w_j, respectively.

$$w_i^T w_j = \log P(j|i)^*$$

$w_i^T w_j$: Similarity b/w i & j

$\log P(j|i)^*$: How likely is j to occur in the context of i?

→ we try to model the dot product similarity with the log probability of j given i

$$w_i^T w_j = \log \frac{X_{ij}}{X_i} = \log X_{ij} - \log X_i \quad -(1)$$

Similarly, $w_j^T w_i = \log \frac{X_{ij}}{X_j} = \log X_{ij} - \log X_j \quad -(2)$

Adding (1) & (2)

$$2 w_i^T w_j = 2 \log X_{ij} - \log X_i - \log X_j$$

$$\Rightarrow w_i^T w_j = \log X_{ij} - \frac{1}{2} \log X_i - \frac{1}{2} \log X_j$$

These 2 terms are dependent on the words itself & not the co-occurrence. Hence, when we model this as a neural network, we can consider these 2 terms as a kind of bias associated with the words i & j. They will still be learnable parameters.

$$w_i^T w_j = \log X_{ij} - b_i - b_j$$

$$\Rightarrow w_i^T w_j + b_i + b_j = \log X_{ij}$$

where w_i, w_j, b_i, b_j are learnable parameters

Loss funct^n: $\min_{w_i\, w_j\, b_i\, b_j} \sum_{ij} \left(w_i^T w_j + b_i + b_j - \log X_{ij}\right)^2$

↳ minimize the dist b/w the 2 terms

Problem: Gives equal weightage to every co-occurrence Ideally, rare & very frequent co-occurrences should have lesser weightage.

Modificat^n: Add a weighting funct^n $f(x)$

Modified loss funct^n: $\min_{w_i\, w_j\, b_i\, b_j} \sum_{ij} f(X_{ij}) \left(w_i^T w_j + b_i + b_j - \log X_{ij}\right)^2$

↳ Penalize very high occurrence

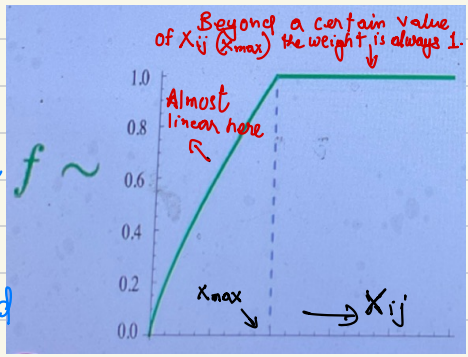
Weighting funct^n

We choose an f such that:

① $f(0) = 0$. If f is viewed as a continuous funct^n, it should vanish as $x \to 0$ fast enough that the $\lim_{x \to 0} f(x) \log^2 x$ is finite

② $f(x)$ should be non-decreasing so that rare co-occurrences are not overweighted

③ $f(x)$ should be relatively small for large values of x, so that frequent co-occurrences are not overweighted



$$f(x) = \begin{cases} (x/x_{max})^{\alpha} & \text{if } x < x_{max} \\ 1 & \text{if } x > x_{max} \end{cases}$$

α can be chosen empirically for a given dataset

Similar, to Word2Vec, we learn 2 embeddings here:
$w_i \to$ target
$w_j \to$ context

The funct^n: $\min_{w_i, w_j, b_i, b_j} \sum_{ij} \left(w_i^T w_j + b_i + b_j - \log X_{ij} \right)^2$

(predict^n component; count component)

↳ This funct^n captures the essence of both predict^n & count based approaches.

GloVe Advantages

* Fast training

*) Scalable to huge corpora
*) Good performance even with small corpus & small vectors (unlike Word2Vec here)

Details About GloVe

Original paper: https://nlp.stanford.edu/pubs/glove.pdf

Blogs with easy explanations:

- https://medium.com/sciforce/word-vectors-in-natural-language-processing-global-vectors-glove-51339db89639
- https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/?fbclid=IwAR3-pws3-K-Snfk6aqbixdxS8zFf-uuPDJ_0ipb94kWeygrdCSEqE9HWmNs
- https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010

*) Word embeddings are not just used for downstream tasks but for various studies as well. eg: word analogy test
Beijing → China
New Delhi → ??
↳ returns India

*) Such models also have bias: Father → Computer Programmer
Mother → ?? → returns homemaker

Man → Doctor
Woman → ?? → returns nurse

Some funny Experiments
*) Take historical data from 1950s to 2010s (like NY times articles)
*) for each decade like 1950-60, 60-70, etc. run GloVe embedding seperately.
*) If you observe the results carefully, you will find that the meaning of words changes across periods (can be verified by looking at the neighbors). eg: broadcast was used earlier for spreading seeds in agriculture but now used for signals, etc.

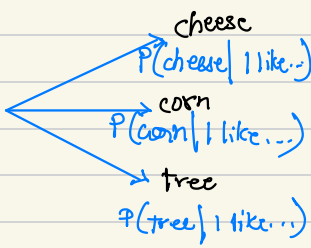
Seperately trained embeddings (or train/update embeddings on-the-fly) can be used as we perform language modeling using Neural Nets.

Lecture 5-1 - Neural Language Models : RNN

Recall : Language modeling

* Language modeling is the task of predicting what word comes next.

I like pizza with loads of _____



* It can be seen as a system that assigns a probability to a text

eg: if we have some text $x_1, x_2 \cdots x_T$, then the probability of this text is:

$$P(x_1, x_2 \cdots x_T) = P(x_1) \cdot P(x_2|x_1) \cdot P(x_3|x_2, x_1) \cdots P(x_T|x_{T-1} \cdots x_1)$$

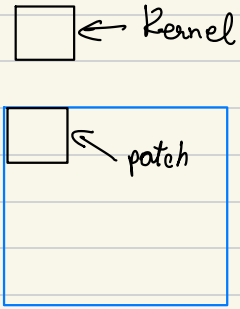
$$= \prod_{t=1}^{T} P(x_T|x_{T-1} \cdots x_1)$$

How to build a Neural Language Model?

* Language Modeling Task:

Input: sequence of words: $x^{(1)}, x^{(2)}, \cdots x^{(t)}$
Output: probability distribution of the next word $P(x^{(t+1)} | x^{(1)}, x^{(2)} \cdots x^{(t)})$

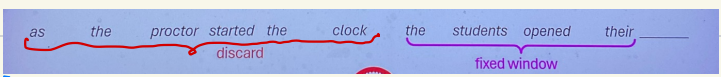
Window based Neural Model (CNN)



How CNN works?
- Identify a patch
- Apply a kernel on the patch
- Aggregate & do max pool, min pool, etc.

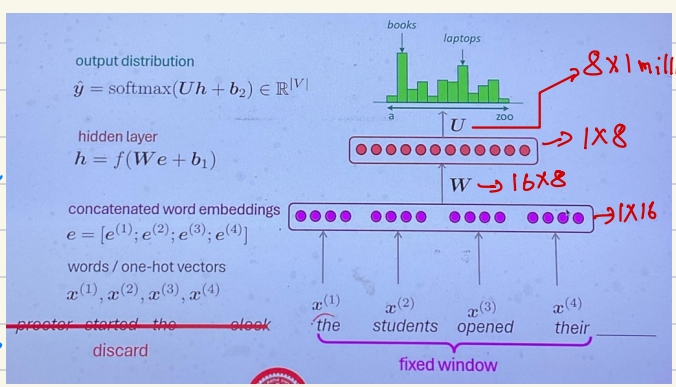
* So, the first thing we do is identify a patch/window size.

* In the case of a language, sentence is a seq. of tokens. The input is 1-D row where tokens are arranged (in the form of word embeddings)



* We will need to identify a kernel/window size. So, if we take a window size of say 4 in the example sentence shown above, then to identify the next word, we only consider the previous 4 words & discard the rest.

* We apply a CNN like layer on this patch of 4 words. If the embedding size is 4, then the patch is a 1X16 matrix. The kernel is applied on top of this which is of the size say 16X8



* On multiplying, we get a vector of size 1X8. This is the hidden state.
* Goal → to predict the next word
  Possible candidates → size of vocab (say 1 million)

* So, we need to up-project from the hidden layer to a vector of size 1 million using a linear layer.
* Then we apply softmax to convert it into a probability distribution & sample a word acc. to the probabilities.
* We expect to sample the words with max probability like books or laptops.

Advantages
- No sparsity problem
- No need to store all observed n-grams

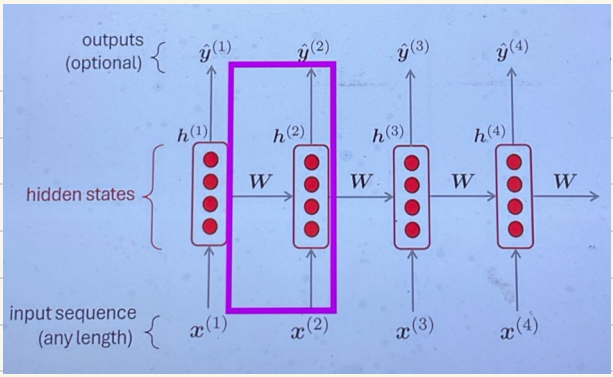
Issues
- Need to fix window size (we have to ignore the rest of the context)
- Window size is a hyperparameter so we need to try multiple values.
- Enlarging the window enlarges the W matrix.
- A final problem is that the embeddings are multiplied with completely different values in W. So there is no symmetry in how the inputs are processed. It is like learning multiple independent params i.e, there is no information passed from the embedding of one vector in a patch to the other

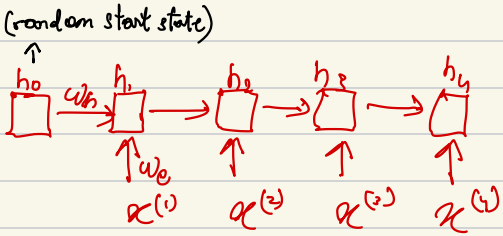
eg: $\begin{bmatrix} 4 & 7 & 1 & 1 \\ 3 & 4 & 2 & 1 \\ 1 & 3 & 7 & 2 \\ 5 & 2 & 5 & 5 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix} = \begin{matrix} 4e_1 + 7e_2 + 1e_3 + 1e_4 \\ 3e_1 + 4e_2 + 2e_3 + 1e_4 \\ 1e_1 + 3e_2 + 7e_3 + 2e_4 \\ 5e_1 + 2e_2 + 5e_3 + 5e_4 \end{matrix}$

Recurrent Neural Networks (RNN)

Core idea: Apply the same weights W repeatedly to the embeddings

* Lets say we have embeddings $x^{(1)}, x^{(2)}, x^{(3)}$ & $x^{(4)}$.
* Each posit^n also has a hidden state associated with it (The position can also be seen as timesteps)
* Hidden states are computed from the input at the current step & hidden state output of the previous step in the following way:





$$h_1 = \alpha(h_0 W_h + x^{(1)} W_e + b)$$

b → bias

α → some non-linear funct^n

$$h_t = \alpha(h_{t-1} W_h + x^{(t)} W_e + b)$$

↳ This is a recurrent funct^n

* The same W_h & W_e are used at each timestep. They are independent of the context size.

Lets say the dim of h is 4. Then the dimension of W_h is 4X4

Similarly, if the dimension of x is 10, then the dimension of W_e will be 4X10 (to match it with the hidden state dimension)

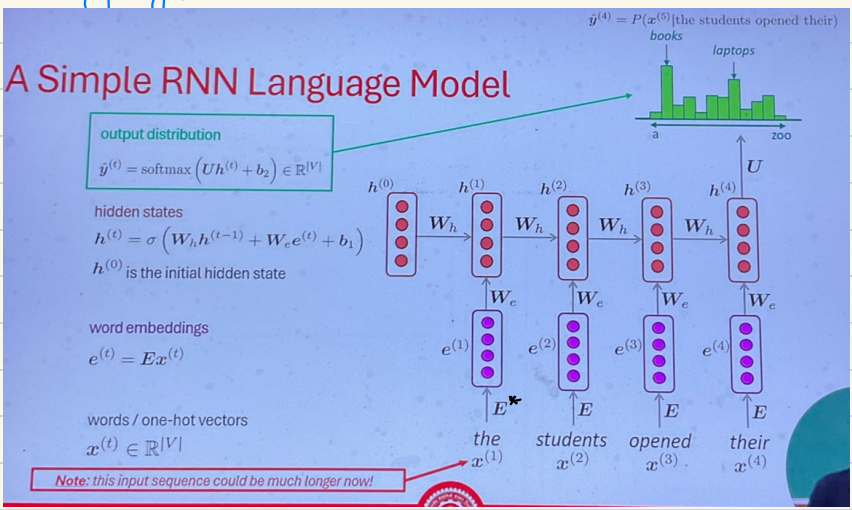
dim of hidden state ↵

↳ dim of embedding

* E is an embedding lookup matrix. Convert a one hot vector of the word to its embeddings using this matrix.

A Simple RNN Language model

* The output of the last layer $h^{(4)}$ is passed through a linear layer U, followed by non-linearity (sigmoid) & then convert it to distribution and sample from it.



Advantages

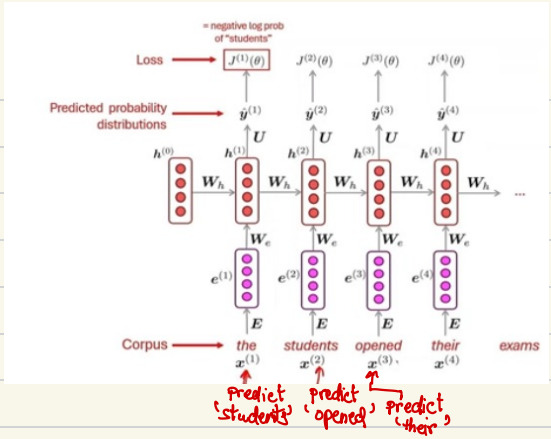
- Can process any length input.
- Computation for step t can (in theory) use information from many timesteps back.
- Model size doesn't increase for longer input context.
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

Disadvantages

- Recurrent computation is slow (to get the output of timestep t we need to calculate all the previous timesteps' output)
- Practically speaking, as the it is difficult to access information from many timesteps back because the effect keeps on decreasing.

Training an RNN

* To train an RNN we need to identify a loss function.



* The diagram alongside shows an RNN. The task is to predict the next word at every timestep.
* So, at the 1st step, we predict 'students', at the 2nd step we predict 'opened' and so on.
* for simplicity, let us assume that the input vectors are one-hot encodings.
* After we generate the hidden state, we pass it through a linear layer followed by some non-linearity i.e., we upsample the output & do a softmax to generate the distribution over vocab words from which we sample.

Lets say that we sample the word 'teachers' at the end of 1st timestep.

So, 'student' is our ground truth & 'teachers' is the sampled word. The loss will be the difference between the embedding of 'student' & the embedding of 'teacher' (i.e., the obtained output). We take this difference using a cross entropy. This cross entropy is calculated at each timestep.

Now, there are 2 paradigms here:

① Ideally, what must happen is that if the model outputs teacher, then the embedding of teacher must be fed in to the next timestep in the RNN and the model should be able to predict 'opened'. If the model predicts 'closed' then 'closed' should be fed to the next timestep in the RNN.

② The 2nd paradigm is that we directly feed the ground truth as input, to the next timestep, irrespective of what output

was obtained as the output in the previous step. This is also called teacher forcing.

Lets say the loss at each timestep was $J^1(\theta), J^2(\theta), J^3(\theta)$, etc. The final loss is the average of all these losses i.e.,

$$J(\theta) = \frac{1}{T}\sum_{i=1}^{T} J^i(\theta)$$

$$J^t(\theta) = \text{Cross Entropy}\left(y^t, \hat{y}^t\right)$$

(y^t: Actual embedding; $\hat{y}^t$: Predicted Embedding)

$$= -\sum_{w \in V} y_w^t \log \hat{y}_w^t = -\log \hat{y}_w^t$$

→ y_w^t corresponding to the ground truth word

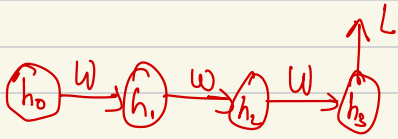
(Because for all other words, the one-hot vector value will be zero)

Q) What are the parameters of this RNN?
A) 3 parameters W_h, W_e & U

Backpropation through time & Multivariable Chain Rule

Lets ignore W_e & U as they're easy to backprop through. Let us consider W_h (referred as W from here on)
Let us assume that we have 4 hidden states for now;

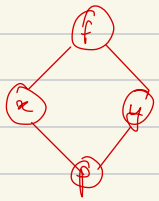
So, how do we calculate $\frac{\partial L}{\partial W}$



We use the concept of multivariable chain rule.

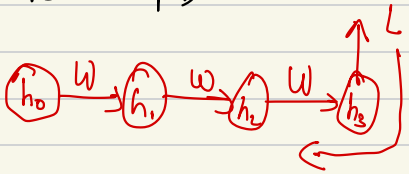
Multivariable Chain Rule: If we have a funct^n $f(x, y)$ such that x & y are functions of p, then to calculate $\frac{\partial f}{\partial p}$

$$\frac{\partial f}{\partial p} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial f}{\partial y}\cdot\frac{\partial y}{\partial p}$$



So, in our case, to calculate $\frac{\partial L}{\partial w}$, we will need to look at the dependence chain.

(for the last node output)

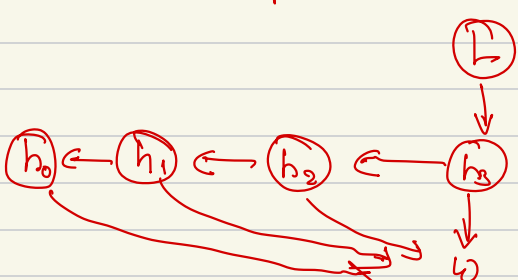


$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h_3}\cdot\frac{\partial h_3}{\partial w}$$

But h_3 is dependent on h_2, which again depends on w

So to compute $\partial h_3/\partial w$, we need to look at all possible paths using multivariable chain rule

Dependence Tree



$$\frac{\partial h_3}{\partial w} = \frac{\partial h_3}{\partial w} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial w} +$$

$$\frac{\partial h_3}{\partial h_2}\cdot\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w} + \frac{\partial h_3}{\partial h_2}\cdot\frac{\partial h_2}{\partial h_1}\cdot\frac{\partial h_1}{\partial h_0}\frac{\partial h_0}{\partial w}$$

$$= \frac{\partial h_3}{\partial w} + \frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial w} + \frac{\partial h_3}{\partial h_1}\frac{\partial h_1}{\partial w} + \frac{\partial h_3}{\partial h_0}\frac{\partial h_0}{\partial w}$$

Q) Can we create a general equat^n!

In reverse order,

$$\frac{\partial h_3}{\partial w} = \frac{\partial h_3}{\partial h_0}\cdot\frac{\partial h_0}{\partial w} + \frac{\partial h_3}{\partial h_1}\cdot\frac{\partial h_1}{\partial w} + \frac{\partial h_3}{\partial h_2}\cdot\frac{\partial h_2}{\partial w} + \frac{\partial h_3}{\partial h_3}\cdot\frac{\partial h_3}{\partial w}$$

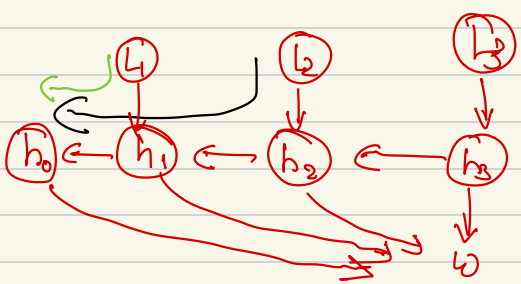
$$\therefore \frac{\partial h_T}{\partial w} = \sum_{k=0}^{T} \frac{\partial h_T}{\partial h_k}\cdot\frac{\partial h_k}{\partial w}$$

$$\therefore \frac{\partial L}{\partial w} = \frac{\partial L}{\partial h_T}\sum_{k=0}^{T} \frac{\partial h_T}{\partial h_k}\frac{\partial h_k}{\partial w}$$

↳ this is the derivative of loss funct^n with backprop through time (BPTT)

Using this, we update the value of w

This is just the loss for the output at the last timestep. There will be similar losses which will be at the 1st, 2nd & 3rd timestep. We need to calculate the derivatives of these losses w.r.t w as well and update w multiple times.



Remember: These are not simple derivatives. h_T & h_k are vectors & w is a matrix

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial h_T}\sum_{k=0}^{T} \frac{\partial h_T}{\partial h_k}\cdot\frac{\partial h_k}{\partial w}$$

Q How to compute $\frac{\partial L}{\partial h_T}$ where $T = 1$?

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial h_3}\frac{\partial h_3}{\partial h_2}\frac{\partial h_2}{\partial h_1}$$

Each of these derivatives creates a problem. ← ↳ Derivative of vector w.r.t vector = matrix

If the value of the matrix is small & we keep on multiplying multiple such small values, we see the final value gets smaller & smaller

Thus, if the matrix is small then the effect of the 1st hidden state on the loss L will be much less than the effect of the last hidden state on loss. This is called the vanishing gradient problem. With increasing depth of the RNN, we keep on forgetting the information we had previously.

Value of matrix = Eigenvalue corresponding to the largest eigenvector.

If it is less than 1, the value of the matrix is small.

Conversely, if the matrix is large (i.e., eigenvalue corresponding to the largest eigenvector is greater than 1), then we have an exploding gradient problem.

NOTE: BPTT has multiple variations like truncated BPTT where we don't allow backprop after certain no. of timesteps

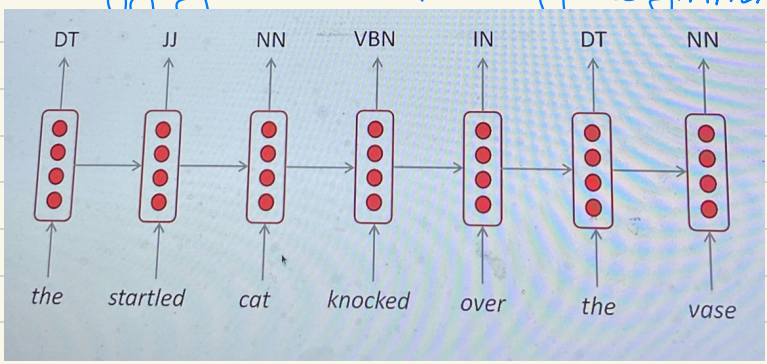
↳ This can be addressed using gradient clipping. At every stage, we can clip the gradient if its value goes beyond a certain threshold.

Vanishing gradient problem can be addressed using advanced RNN methods like LSTM & GRU.

Lecture 5.2 — Neural Language Models: LSTM & GRU

Uses of RNNs

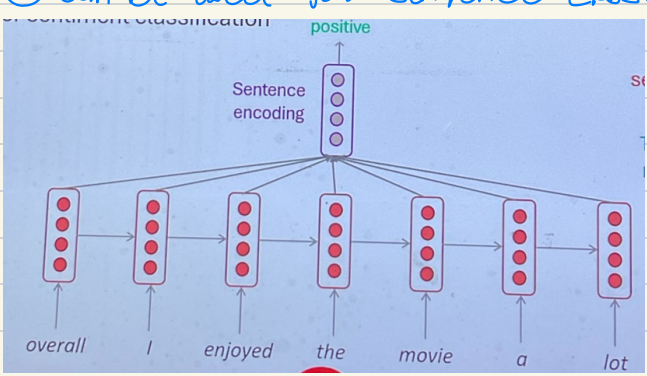
① Can be used for sequence tagging tasks like POS tagging & named entity recognition.



⇒ Note: in such tasks we choose the one with the highest probability (& not sample) because it is not a generation task.

But in vanilla RNNs we only have access to the previous hidden states. But sometimes for NER or POS tagging, we might need information from future tokens as well. To solve this we use bidirectional RNNs i.e., 1 RNN from left to right & 1 from right to left. Each position, thus, has 2 hidden states (one for both directions)

② Can be used for sentence classification tasks like sentiment analysis or spam classification.

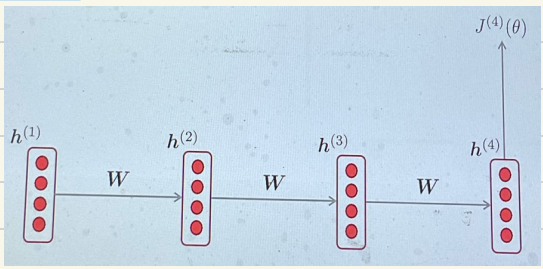


How?

- One way is to merge the outputs of all hidden states & merge them to create a sentence encoding which can be used for the downstream task.

- Another way is to take the output of only the last hidden state (since it contains info^n from all the previous hidden states) and pass it through a linear layer to perform the downstream task.

Vanishing & Exploding Gradients



* Happens because when we have to take the gradient of $J^{(4)}(\theta)$ wrt $h^{(1)}$ then we need to take the gradient of $h^{(4)}$ wrt $h^{(3)}$, $h^{(3)}$ wrt $h^{(2)}$ & so on.

$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial h^{(4)}}{\partial h^{(3)}} \times \frac{\partial J^{(4)}(\theta)}{\partial h^{(4)}}$$

↳ Gradient of vector wrt vector is matrix. If this matrix is small, it leads to vanishing gradients & if large it leads to exploding gradients.

Effect of vanishing gradients on RNN-LM

* **LM Task:** When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____

* To learn from this training example, the RNN-LM needs to model the dependency b/w "tickets" on the 7th step and the target word "tickets" at the end.

* But if the gradient is small, the model can't learn this dependency
  - So, the model is unable to predict similar long distance dependencies at test time.

How to fix the vanishing gradient problem?

* The main problem here is that it's too difficult for the RNN to learn to preserve information over many timesteps.
* In a vanilla RNN, the hidden state is constantly being rewritten

$$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_x x^{(t)} + b\right)$$

* What if we have an RNN with seperate memory which is added to?
  LSTMs & GRUs use gates to do just that

* And then create more direct and linear pass-through connections in the model.
  - like attention, residual connections, etc.

Long Short Term Memory (LSTM)

* A type of RNN proposed by Hochreiter & Schmidhuber in 1997 as a solut^n to the vanishing gradients problem.

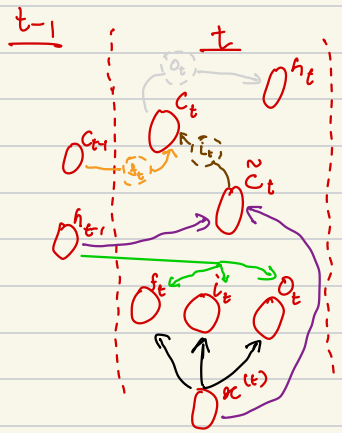
* On step t, there is a hidden state $h^{(t)}$ and cell state $c^{(t)}$
  - Both are vectors of length n
  - The cell stores long-term information.

- The LSTM can erase, write & read information from the cell.

* The selection of which information is erased/written/read is controlled by three corresponding gates (gates are calculated things whose values are probabilities)
  - Gates are also vectors of length n
  - On each timestep, each element of the gate can be open (1), closed (0) or somewhere in between.
  - The gates are dynamic i.e., their value is computed based on the current context.

LSTM has 3 types of gates:
1. Input Gate (i_t)
2. Forget Gate (f_t)
3. Output Gate (O_t)

* We have a sequence of inputs $x^{(t)}$, and we will compute a sequence of hidden states $h^{(t)}$ and cell states $c^{(t)}$. On timestep t:



controls what is kept & what is forgotten from the previous hidden state. Thus, f is attached to $c^{(t-1)}$ i.e., how much to move from $c^{(t-1)}$ to $c^{(t)}$

$$f^{(t)} = \sigma\left(W_f h^{(t-1)} + U_f x^{(t)} + b_f\right)$$

controls what parts of the new cell content are written to the cell. Thus, i is attached to $\tilde{c}^{(t)}$ i.e., the new content generated from $h^{(t-1)}$ & $x^{(t)}$

$$i^{(t)} = \sigma\left(W_i h^{(t-1)} + U_i x^{(t)} + b_i\right)$$

controls what part of the cell are output to the hidden state. Thus, O is attached to $c^{(t)}$.

$$O^{(t)} = \sigma\left(W_o h^{(t-1)} + U_o x^{(t)} + b_o\right)$$

→ Sigmoid ensures all values are b/w 0 & 1 and function like probabilities (on gates)

$$\tilde{C}^{(t)} = \tanh\left(W_c h^{(t-1)} + U_c x^{(t)} + b_c\right)$$

$$C^{(t)} = f^{(t)} \odot C^{(t-1)} + i^{(t)} \odot \tilde{C}^{(t)}$$

$$h^{(t)} = O^{(t)} \odot \tanh C^{(t)}$$

→ element-wise multiplication (or Hadamard product)

* $f^{(t)}$, $i^{(t)}$ & $O^{(t)}$ are all functions of the previous hidden state & current input. The difference is in the weights. All 3 gates have 2 sets of weights each, W & U which are learnable parameters (ignoring the bias)

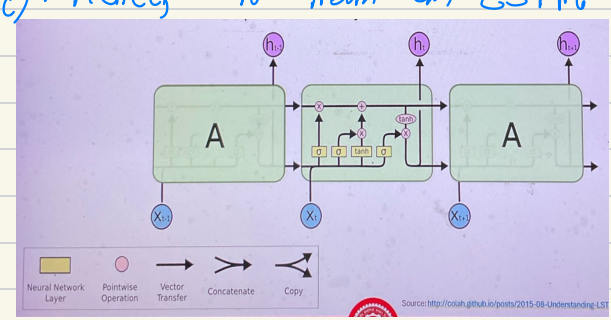
* $\tilde{C}^{(t)}$ is the content that we want to write to the cell

* However, the input & forget gate won't allow us to write this directly to the cell state. $f^{(t)}$ will decide how much of $C^{(t-1)}$ will be written to $C^{(t)}$ & $i^{(t)}$ will decide how much of $\tilde{C}^{(t)}$ will be written to $C^{(t)}$.

* Finally, how much of the cell state would we like to read & write to the hidden state will be decided by $O^{(t)}$

Issues

* As compared to 3 sets of weights for vanilla RNN (W_e, W_h & V) we have 8 set of weights in an LSTM ($W_f, U_f, W_i, U_i, W_o, U_o, W_c, U_c$). Hence, to train an LSTM network, we need a lot of data or we run the risk of overfitting.



How does LSTM solve the vanishing gradient problem?

* LSTM architecture makes it much easier for an RNN to preserve information over many timesteps.
  - For eg, if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely.
  - In contrast, it's harder for a vanilla RNN to learn a recurrent weight matrix W_h that preserves info in the hidden state.
  - In practice, you get about 100 timesteps rather than about 7.
* LSTM doesn't guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies.
* There are also alternative ways of creating more direct & linear pass through connections in models through long distance dependencies.

Is vanishing gradient just an RNN problem?

* No! It can be a problem for all neural architectures (including feed forward & convolutional neural networks), especially deep ones
  - Due to chain rule & choice of non-linearity function, gradient can become vanishingly small as it backpropagates.
  - Thus, lower layers are learned very slowly (i.e., hard to train)
* Another solution: lots of new feed forward/convolutional architectures add more direct connections (thus allowing the gradient to flow)

eg:

- Residual connections aka "ResNet"
- Also known as skip connections
- The identity connection preserves information by default
- This makes deep networks much easier to train.

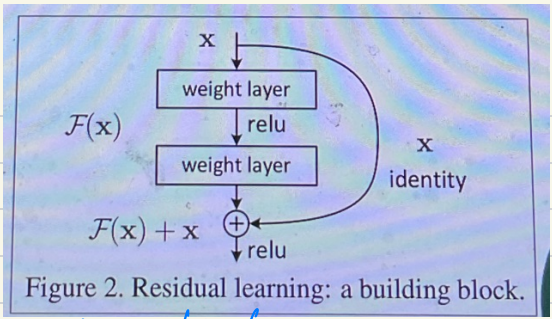


Figure 2. Residual learning: a building block.

Other Methods:

- Dense connections aka "DenseNet"
- Directly connect each layer to all future layers!

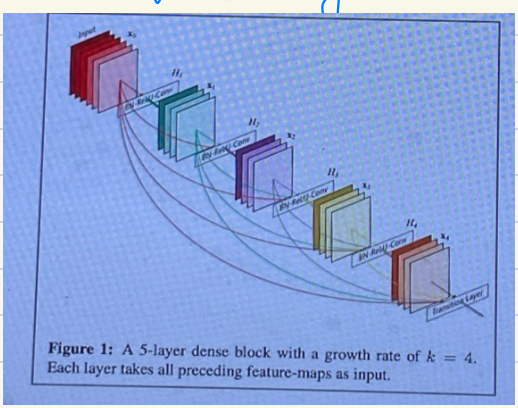
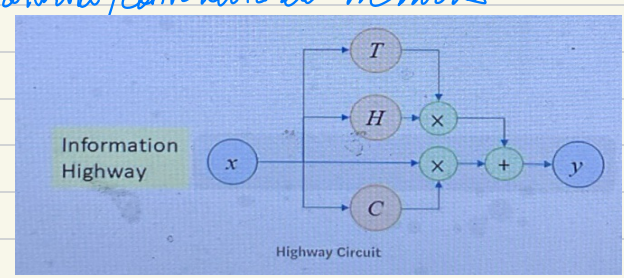


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

- Highway connections aka "Highway Net"
- Similar to residual connections, but the identity connection vs the transformation layer is controlled by a dynamic gate.
- Inspired by LSTMs but applied to deep feed-forward/convolutional networks



Conclusion: Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to repeated multiplication by the same weight matrix.

Gated Recurrent Units (GRUs)

* Proposed by Cho et al in 2014
* On each timestep t, we have input $x^{(t)}$ and hidden state $h^{(t)}$ (no cell state)
* 2 gates: ① Update gate ② Reset gate

Update gate: controls what part of the hidden state are updated vs preserved
Hence, attached to $h^{(t-1)}$ & $\tilde{h}^{(t)}$

$$u^{(t)} = \sigma(W_u h^{(t-1)} + U_u x^{(t)} + b_u)$$

Reset gate: controls what part of the previous hidden state are used to compute new content
Hence, attached to $h^{(t-1)}$

$$r^{(t)} = \sigma(W_r h^{(t-1)} + U_r x^{(t)} + b_r)$$

New hidden state content: reset gate selects useful parts of previous hidden state. Use this & current input to compute new hidden content

$$\tilde{h}^{(t)} = \tanh(W_h (r^{(t)} \circ h^{(t-1)}) + U_h x^{(t)} + b_h)$$

Hidden state: update gate simultaneously controls what is kept from previous hidden state & what is updated to new hidden state content.

$$h^{(t)} = (1 - u^{(t)}) \circ h^{(t-1)} + u^{(t)} \circ \tilde{h}^{(t)}$$

* Both $u^{(t)}$ & $r^{(t)}$ are functions of $h^{(t-1)}$ & $x^{(t)}$ similar to LSTMs.
* There is no cell content, but there's a hidden cell content ($\tilde{h}^{(t)}$) that we want to insert into the hidden state.
* $\tilde{h}^{(t)}$ is controlled by $r^{(t)}$ which decides how much of the previous hidden state will be used to generate the content
* Then content is then controlled by the update gate (i.e., it controls how much of the current (hidden) cell content will be moved to the current hidden state & how much of the previous hidden state will be preserved)

* There are 6 parameter sets here ($W_u, U_u, W_r, U_r, W_h, U_h$)

Q) How does it solve vanishing gradient?
A) Like LSTM, GRU makes it easier to retain info for long-term (eg: by setting update gate to 0)

LSTM vs GRU

* Researchers have proposed many gated RNN variants, but LSTM & GRUs are most widely used.
* The biggest difference is that GRU is quicker to compute and has fewer parameters.
* There is no conclusive evidence that one consistently performs better than the other.
* LSTM is a good default choice (especially if you have particularly long dependencies, or lots of training data)
* Thumb rule: start with LSTM, but switch to GRU if you want something more efficient.

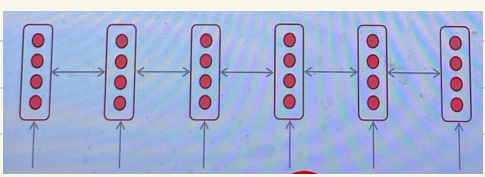
Bidirectional RNNs

On timestep t,

Forward RNN $\overrightarrow{h}^{(t)} = RNN_{FW}(\overrightarrow{h}^{(t-1)}, x^{(t)})$

Backward RNN $\overleftarrow{h}^{(t)} = RNN_{BW}(\overleftarrow{h}^{(t+1)}, x^{(t)})$

Concatenated hidden states: $h^{(t)} = [\overrightarrow{h}^{(t)}; \overleftarrow{h}^{(t)}]$



Generally, these 2 RNNs have separate weights

Multilayer RNN

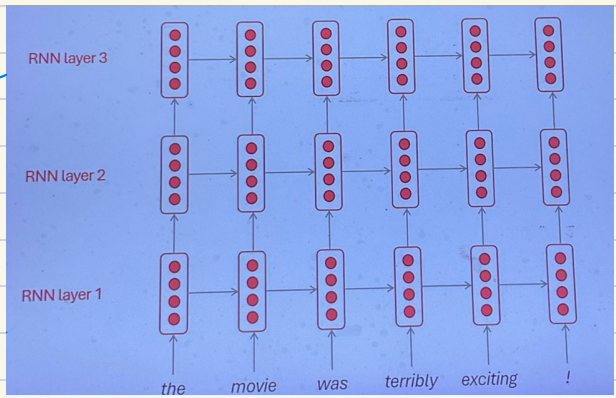
* RNNs are already "deep" on one dimension (they unroll over many timesteps)
* We can also make them "deep" in another dimension by applying multiple RNNs - this is a multi-layer RNN.
* This allows the network to learn more complex representations. The lower RNNs should compute lower level features and the higher RNNs should compute higher level features.
* Multilayer RNNs are also called stacked RNNs.

The hidden states from RNN layer i are inputs to RNN layer i+1

Multi layer vs bidirectional

↳ Bidirectional generates current hidden state from the current input $x^{(t)}$, previous hidden state $h^{(t-1)}$ of forward RNNs & previous hidden state $h^{(t+1)}$ of backward RNN.

↳ In multi layer RNN, the inputs to the higher layer is no longer the true input but the output of the previous layer's hidden state.



LSTMs: Real world success

* In 2013-2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include handwriting recognition, speech recognition, machine translation, parsing & image captioning, as well as language models
  - LSTMs became the dominant approach for most NLP tasks.
* Now (2019-2024), transformers have become dominant for all tasks
* But RNNs are still very important (and making a resurgence? eg: Mamba)
* But one problem remains - we cannot process the sequence in parallel. (Update - "Were RNNs all we needed?" a paper published in Oct 2024 tries to address this problem by removing the hidden state dependencies from the inputs of LSTMs & GRUs, thereby enabling parallelization. Introduces min LSTM & min GRU - Explore?)

Lecture 5.3 – Neural Language Models: Seq2Seq & Attention

Neural Machine Translation

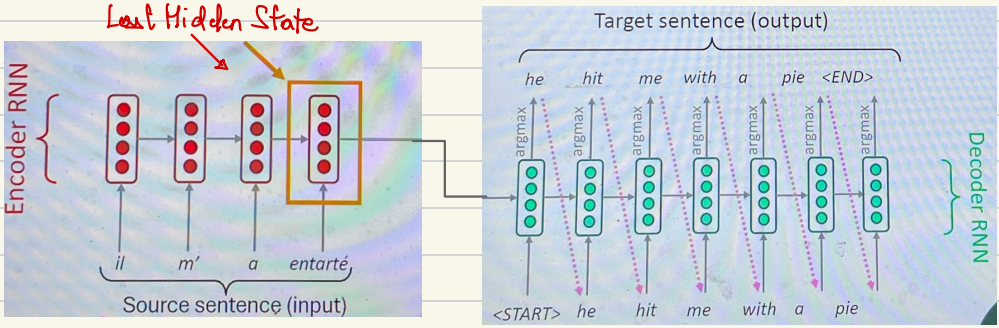
* Neural Machine Translat^{n} (NMT) is a way to do machine translat^{n} with a single neural network
* The neural network architecture is called sequence-to-sequence (seq2seq) [as it takes a sequence as input & returns a sequence as output] and it involves 2 RNNs (encoder & decoder).

Many-to-Many ⟶ Summarisat^{n}, dialogue, etc.
Many-to-One ⟶ Classificat^{n} (not really since it outputs label & not sequence)
One-to-Many ⟶ Image Captioning

The Sequence-to-Sequence Model

Encoding of the source sentence. Provides initial hidden state for Decoder RNN

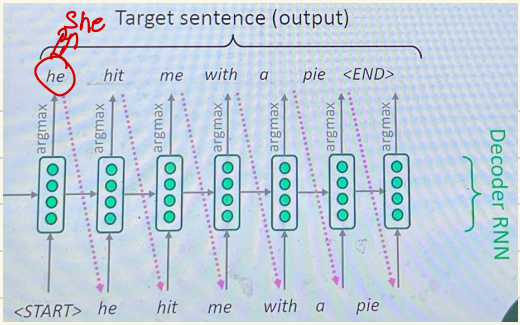
NOTE: Diagram shows test time behavior: decoder output is fed as next step input



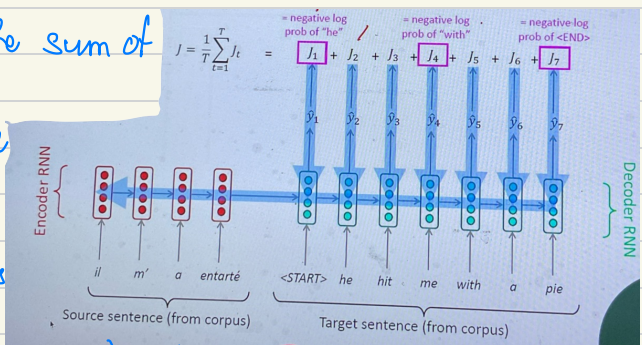
* Encoder RNN produces an encoding of the source sentence.
* The loss is computed at the decoder RNN.

Recall: Teacher Forcing

- Ideally, if the model generates "she" then "she" should be fed into the next step input
- But in teacher forcing, we feed the original input "he" itself. (Kind of cheating but turned out to be very effective when training the model)



* Total error $J = \frac{1}{T}\sum_{t=1}^{T} J_t$ is the sum of individual errors normalized by the no. of timesteps.



* During backpropagation (BPTT), this loss 'J' will affect all the hidden states (including encoder). This is called end-to-end training.

* Seq2Seq model is an example of a conditional language model.
  - Language Model: because the decoder is predicting the next word on the target sentence y
  - Conditional: because its predictions are also conditioned on the source sentence X.

* NMT directly calculates $P(y|x)$

$$P(y|x) = P(y_1|x)\, P(y_2|x, y_1)\, P(y_3|x, y_1, y_2) \cdots P(y_T|x, y_1, y_2 \cdots y_{T-1})$$

Probability of the next target word, given target words so far and source sentence X.

Q) How to train an NMT system?
Do teacher forcing, compute loss at each timestep, BPTT (end-to-end), repeat for all pairs of sentences

Q) How many weight matrices?
A) W_e & W_h for encoder → NOTE: No U matrix here because were not predicting anything at each timestep.
W_e', W_h' & U for decoder
↳ All these matrices are learned

NOTE: Embeddings will be different for tasks like language translation but will be same for tasks like summarization.

Inference

→ The sentence is passed through the encoder RNN,

→ the final hidden state output is calculated & passed to the decoder,

→ the decoder takes in the output of last hidden state of encoder as the initial hidden state input (h_0) and the <START> token as initial input

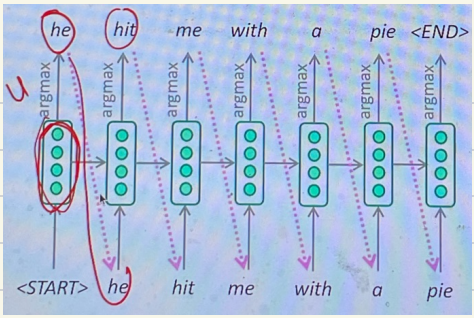
→ The first hidden state output is calculated, passed through a linear layer U which projects it to a higher dimension (i.e., dimension of the vocabulary) followed by a non-linearity (softmax)

→ finally we sample one token from the vocabulary & hope that the token sampled is the one with highest probability.

Greedy Decoding

*) Lets say we sample 'he' & feed it to the next timestep (i.e., feed the embedding of 'he' as input to next timestep)

*) In a greedy decoding strategy, instead of sampling the token based on the probabilities, we choose the token which has the highest probability.



Issue
↳ No way to undo decisions
eg: Input : il a m'entarté (he hit me with a pie)
↳ he ____
↳ he hit ____
↳ he hit a ____ (no going back now, because you never get a diff. argmax)

Q) How to address this issue?

Exhaustive Search Decoding

*) What if we do some exhaustive search to address this problem?
Instead of generating 1 token at a time, if we generate V tokens (V = vocab size), pass it to the next layer, generate V tokens again & so on, such that we maximize:

$$P(y|x) = P(y_1|x) \cdot P(y_2|x, y_1) \cdots P(y_t|x, y_1 \cdots y_{t-1})$$
$$= \prod_{i=1}^{T} P(y_t|x, y_1, y_2 \cdots y_{t-1})$$

*) The time complexity at each timestep will be V^t at each timestep (or $O(V^T)$) → way too expensive

Beam Search Decoding

Q) Doing an exhaustive search is not possible but what if we try to squeeze the search space?

Core idea: Similar to exhaustive search but instead of keeping track of all words in the vocab on each step of the decoder, we keep track of 'k' most probable partial translations (which we call hypotheses)

↳ 'k' is called the beam size (5 or 10 mostly in practice)

* A hypothesis $y_1 \cdots y_t$ has a score which is its log probability

$$\text{score}(y_1 \cdots y_t) = \log P_{LM}(y_1 \cdots y_t | x) = \sum_{i=1}^{t} \log P_{LM}(y_i | x, y_1 \cdots y_{i-1})$$

- Scores are all -ve, and higher score is better
- We search for high scoring hypotheses, tracking top k on each step

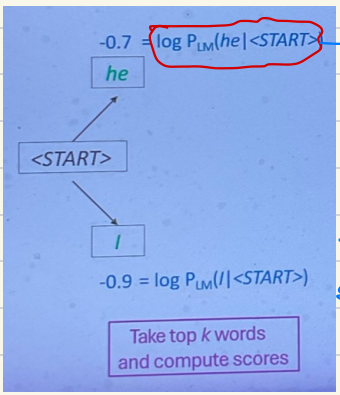
* Beam search is not guaranteed to find optimal solut^n But much more efficient than exhaustive search.

Complexity = $O(k^T)$ → still high but much less than beam search

Q) When do we stop decoding?
A) When <EOD> token is generated

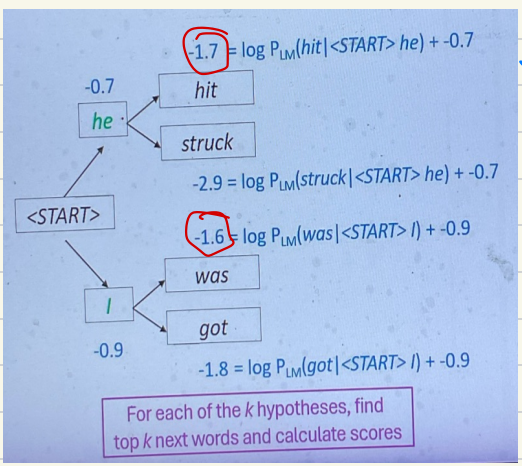
Example

Beam size $k = 2$ $\left[score(y_1 \cdots y_t) = \sum_{i=1}^{T} \log P_{LM}(y_i | y_1 \cdots y_{i-1}, x) \right]$



→ Obtained from the probability distribution generated by projecting the hidden layer output to a higher dimension using a linear layer U & softmax.

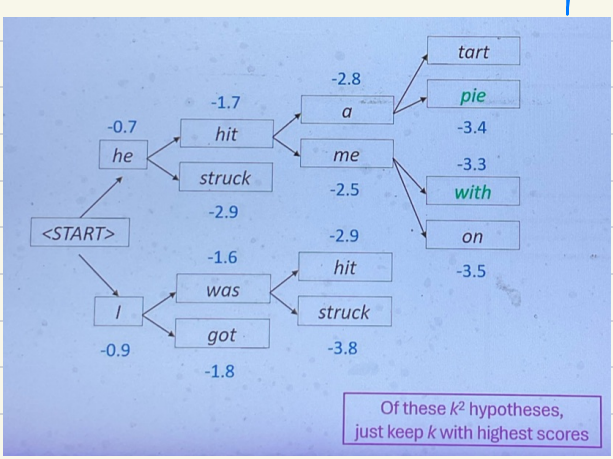
→ We choose the 2 words with highest probability (Remember, numbers are b/w 0 & 1 so the log will be -ve but we need to choose the ones with the highest value)
In this case, we choose 'I' & 'he'.

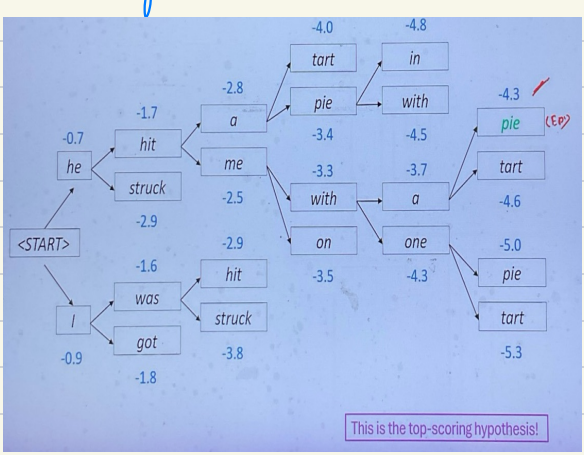


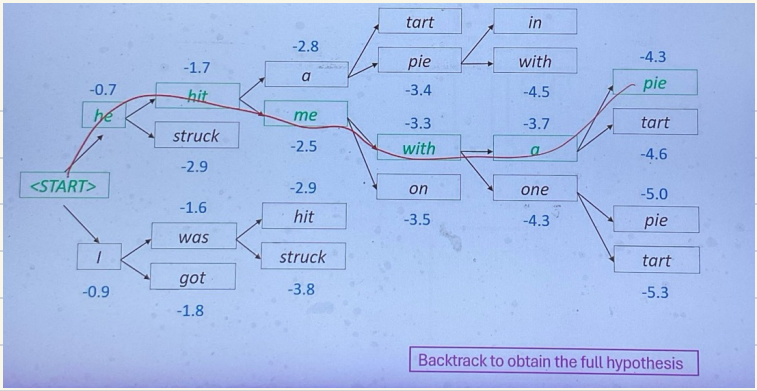
→ Now, for 'I' & 'he' we get 2 words each.

→ We calculate the total probability for all 4 and choose the 2 with highest probability i.e., 'he hit' & 'I was'

→ We discard the other paths as we want only top 2 paths per timestep.







Issue
- Longer hypotheses will be penalized & shorter ones will be preferred because as we keep adding -ve numbers, then the total will always reduce

(or we can say that the product of multiple probabilities will keep getting smaller & smaller)

To mitigate this, we normalize the score by the no of tokens

$$\frac{1}{t}\sum_{i=1}^{t} \log P_{LM}(y_i | y_1, \cdots y_{i-1}, x)$$

NMT: The first Big Success Story of NLP Deep Learning

* NMT went from fringe research attempt in 2014 to the leading standard method in 2016.
  - 2014: 1st seq2seq paper published [Sutskever et al]
  - 2016: Google switches from Statistical MT to Neural MT (by 2018 everyone had)

Issues with RNN

- Linear interaction distance (t_3, t_4 is treated the same as t_5, t_6) (token 3)
- Bottleneck problem
- Lack of parallelizability (You need t_1 to compute t_5)

Bottleneck problem - Decoder has access to only the last hidden state & not the intermediate ones. So, the final hidden state output must remember critical information from all the

tokens (or timesteps) in the input. So, there is a bottleneck here in terms of remembering a lot of information.

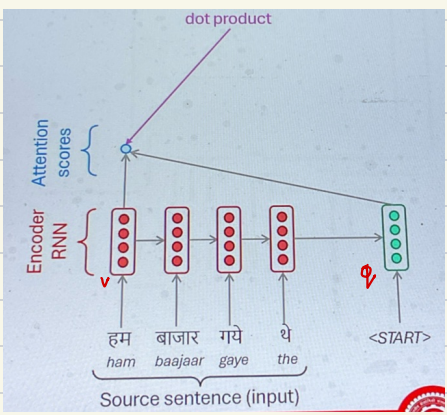
Attention

*) Provides a solution to the bottleneck problem.

*) Core idea: on each step of the decoder, use direct connection to the encoder to focus on a particular part of the sequence

Lets visualize the attention mechanism

*) The 1st hidden state of the decoder has access to all the previous encoder states

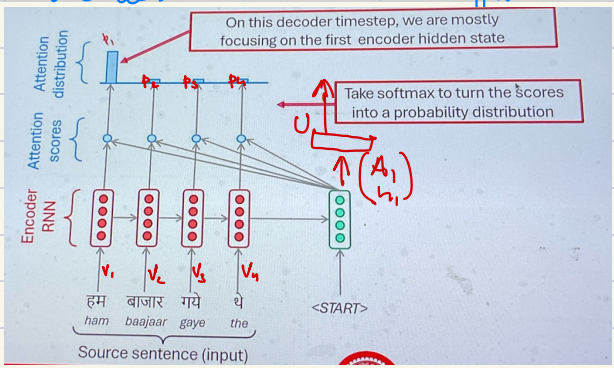


Q) How to do this?

- We introduce the concept of query & value.
- Each of the decoder hidden states will act as a query & encoder hidden states will act as a value.
- Each decoder hidden state will check, which of the encoder hidden states is relevant for that decoder hidden state i.e., which encoder state will be helpful for the decoder to generate the correct output of that decoder state.
- Computationally, we first measure the simi/arity b/w query(q) & value(v) vectors using dot product to obtain a scalar value.
- Each decoder hidden state output will be used to calculate

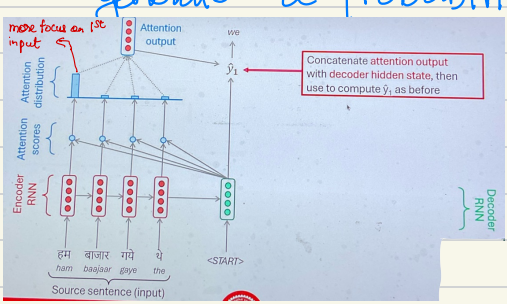
a dot product with each encoder hidden state output.

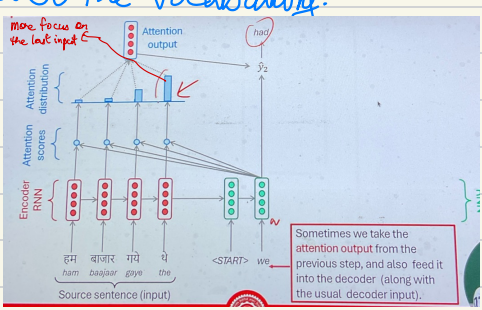
- These dot products are scalar values that indicate the similarity between the two vectors. They are called attention scores
- We pass it through softmax which will produce a distribution called attention distribution.
- From these values (1 & p), we generate the attention vector which is the weighted sum of encoder states (the weights being the probabilities in the attention distribution)

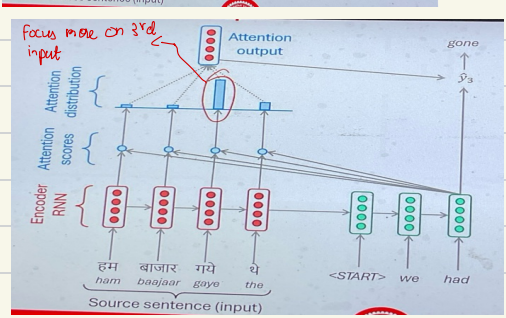


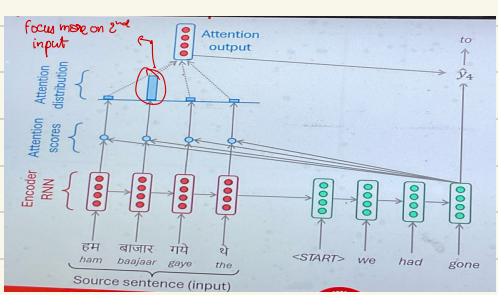
$\therefore$ Attention vector $(A) = p_1 v_1 + p_2 v_2 + p_3 v_3 + p_4 v_4$

- Attention vector & decoder hidden state are concatenated, passed through a linear layer (U) followed by non-linearity to generate a probability over the vocabulary.









* No bottleneck problem here as there is a connection from every decoder state to every encoder state.

Q) Did we use any additional parameters?
A) No, we still have the same params W_h, W_e, W_h' & W_e' & U.

Attention: In Equations

* We have encoder hidden states $h_1 \ldots h_N \in \mathbb{R}^h$
* On timestep t, we have decoder hidden state $s_t \in \mathbb{R}^h$
* We get the attention scores e^t for this step:

$$e^t = [s_t^T \cdot h_1, \ldots, s_t^T h_N] \in \mathbb{R}^N$$

* We take softmax to get the attention distribut^n α^t for this step (this is a probability distribut^n, sums to 1)

$$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^N$$

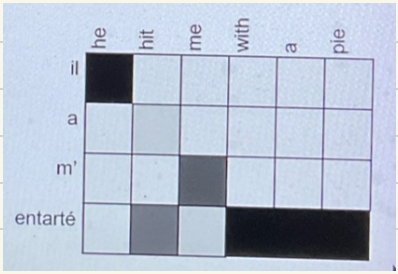
* We use α^t to take a weighted sum of the encoder hidden states to get the attention output a_t

$$a_t = \sum \alpha_i^t h_i \in \mathbb{R}^h$$

* Finally, we concatenate the attention output a_t with the decoder hidden state s_t and proceed as in the non-attention seq2seq model

$$[a_t ; s_t] \in \mathbb{R}^{2h}$$

Attention is great



*) Significantly improves NMT performance
↳ Allows decoder to focus on certain parts of the source

*) Solves the bottleneck problem
↳ Allows decoder to peek directly at source; bypass bottleneck

*) Helps with vanishing gradient problem.
↳ Provides shortcut to faraway states

*) Provides some interpretability
↳ Inspecting attention distribution allows us to see what the decoder was focusing on.
↳ We get (soft) alignment for free which is cool because we never trained an alignment system. The network learned alignment by itself.

* A statistical machine translation model has 2 components - ① translation ② alignment (i.e., re-arrangement) [eg: 'मैं बाज़ार जा रहा हूँ' does not translate directly to 'I am going to the market' word-by-word. There is some re-arranging - for alignment, we need a seperate model but here we get it for free (NOTE: Attention interpretability is still debated)

Attention is a 'General' Deep Learning Technique

*) Attention is great for improving seq2seq model for Machine Translation task but can be used in many architectures (not just seq2seq) and many tasks (not just MT)

*) More general definition of attention:

- Given a set of vector values and vector query, attention is a technique to compute the weighted sum of the values, dependent on the query.

*) We sometimes say that the query attends to the values.
*) for eg, in seq2seq + attention model, each decoder hidden state (query) attends to all encoder hidden states (values)

*) Intuition:
- The weighted sum is a selective summary of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a fixed size representation of an arbitrary set of representations (the values), dependent on some other representation (the query).

Variants of Attention

- Original formulation: $a(\mathbf{q}, \mathbf{k}) = w_2^T \tanh(W_1[\mathbf{q}; \mathbf{k}])$
- Bilinear product: $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T W \mathbf{k}$ Luong et al., 2015
- Dot product: $a(\mathbf{q}, \mathbf{k}) = \mathbf{q}^T \mathbf{k}$ → discussed here Luong et al., 2015
- Scaled dot product: $a(\mathbf{q}, \mathbf{k}) = \frac{\mathbf{q}^T \mathbf{k}}{\sqrt{|\mathbf{k}|}}$ Vaswani et al., 2017

More information:

*) We can also think of parameterized attention mechanisms if we have enough data.

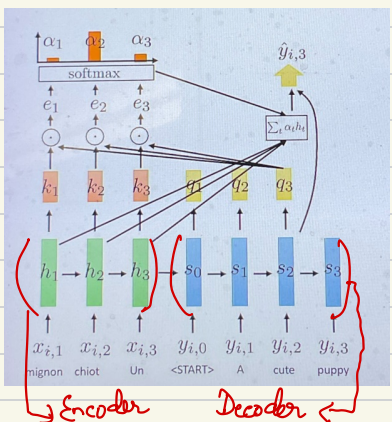
Lecture 6.1 – Intro to Transformer: Self & Multihead Attention

Is attention all we need?

*) Attention mechanism discussed above still doesn't address the issue of sequential process & lack of parallelisation.

Recap: Attention

*) for Machine translat^n tasks,
$q_3 k_1$, $q_3 k_2$ $q_3 k_2$



where q & k are the hidden states of decoder & encoder respectively (called query & value previously)

*) Then we obtain the similarity scores, convert it to a probability distribut^n (by applying softmax) & generate the attent^n vector from it.

*) Now, we call them as query & key (we introduce a new concept of query, key & value)

*) Similar to information retrieval [what do we compare a query in a search engine with, to retrieve relevant documents?
Ans: Keywords, document metadata, etc.]

*) In a vanilla, attention RNN setup, the hidden states themselves are the q & k vectors. We do the dot product followed by softmax to obtain the attention distribut^n

*) Now, we also have a value vector (same as the encoder hidden state h_i for vanilla attent^n RNN) which we multiply with the probabilities obtained (which act as weights) &

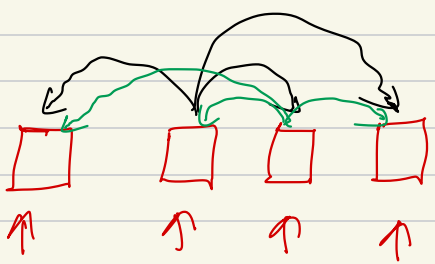
sum it all up to obtain the attent^{n} vector. Same process is repeated for all decoder states.

Q) If through attent^{n} all decoder states can access each encoder state, then why do we need recurrent connections? Can we transform our RNN into a purely attent^{n} based model?

A) Yes, through self-attention

Self-Attention

*) In self-attent^{n}, we remove the dependencies of recurrent connections. Now, the hidden states look like in the diagram below



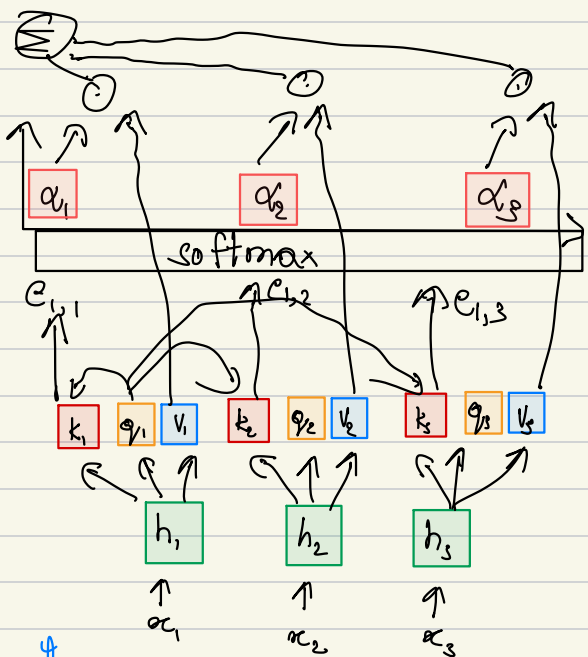
*) Each state has access to all the other states & each operat^{n} is independent of each other i.e., all states can be computed together. This is the premise behind self-attent^{n}.

Issues

- When we discard the recurrent connections, we lose out on the informat^{n} of the sequence (eg: when we shuffle the hidden states, we get the same output. So,

The order of the sequence has no significance here. The temporal dependency preserved in RNNs is lost)

Self-Attention Mechanism



* Inputs are passed through a linear layer followed by non-linearity to obtain h_1, h_2 & h_3.

$$h_t = \sigma(Wx_t + b)$$

weights are shared ↵ even though it is not a recurrent model

* From each hidden state we generate 3 vectors - query, key & value using 3 parameter matrices - W_q, W_k & W_v

$$q_t = W_q h_t \qquad k_t = W_k h_t \qquad v_t = W_v h_t$$

↳ These weights are shared as well

* The attention process is repeated as before: for each token, we take the query vector q_t and multiply it with the keys of every other token to obtain the attention scores.

$$e_{i,t} = q_i \cdot k_t$$

query ↵ ↳ key

* The scores are passed through a softmax layer to obtain the attention distribution.

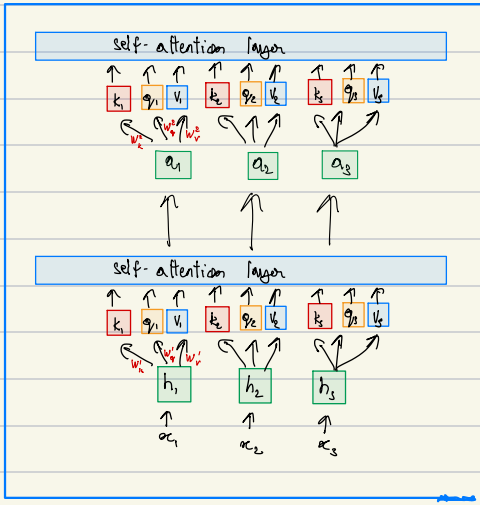
$$\alpha_{t,t'} = \exp(e_{t,t'}) \Big/ \sum_{t''} \exp(e_{t,t''})$$

* Finally, we compute the attention vector by multiplying the value vector of each token with its corresponding probability and taking the sum.

$$a_t = \sum_{t'} \alpha_{t,t'} v_{t'}$$

* Similarly, the attention vectors for all tokens can be generated independently, irrespective of position.
* All the steps, described above, from q, k, v computation to attention vector calculation can be thought of as one single layer.

* We can stack multiple self-attention layers on top of one another.
* For each layer, the W_q, W_k & W_v matrices will be different
* The more data we have, the more attention layers we can stack.
* Self-attention allows access of tokens in parallel, thus allowing us to use the parallelisation capabilities of modern GPUs



From self-attention to Transformers

* Transformers is a class of models that doesn't use recurrent

connections but relies entirely on attentⁿ for processing sequences.

* But, we need to address a few key limitations:
  1) Positional Encoding - addresses lack of sequence informatⁿ
  2) Multi-headed attention - allows querying multiple positions at each layer
  3) Adding non-linearities - so far, each successive layer is linear in the previous one
  4) Masked decoding - how to prevent attention lookups into the future?

Positional Encoding

1) Problem: Self-attentⁿ processes all the elements of a sequence in parallel w/o any regard for their order.
   - eg: the sun rises in the east
     the east rises in the sun
   - Self-attentⁿ is permutation invariant.
   - In natural language, it is important to take the order of words in a sentence.

Solution: Explicitly add positional informatⁿ to indicate where a word appears in a sequence.

How?

- Let's say input is set of words: x_1, x_2, x_3, x_4
- Embeddings: e_1, e_2, e_3, e_4 (one-hot, Word2Vec, GloVe, etc.)
- for each one get the positional encoding: p_1, p_2, p_3, p_4
- Add them (i.e., $e_1+p_1, e_2+p_2, e_3+p_3, e_4+p_4$) & feed it to self-attentⁿ block

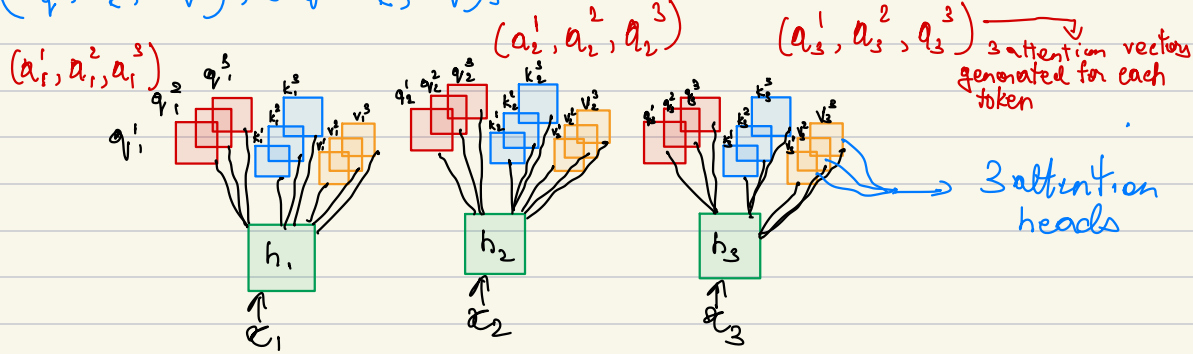
How to calculate positional encoding is discussed later?

NOTE: e_i & p_i are both vectors. Positional informatⁿ is encoded into vectors.

Multiheaded Attention

* Given that we are fully dependent on attention now, it could be beneficial to include more than 1 time step.

* So, instead of just one set of W_q, W_k, W_v parameters we have multiple such matrices in multi-head attention like (W_q^1, W_k^1, W_v^1), (W_q^2, W_k^2, W_v^2), etc.



* From each parameter set, we obtain a new set of query, key & value vectors for each token (q_1^1, k_1^1, v_1^1), (q_1^2, k_1^2, v_1^2), etc.
* Each set of parameter matrices is called an attention head.
* Through each attention head we generate an attention vector - $(a_1^1\ a_1^2\ a_1^3)$, $(a_2^1\ a_2^2\ a_2^3)$ $(a_3^1\ a_3^2\ a_3^3)$
* Now we have 3 attention vectors for each token i.e., 1 for each head

Q) Why multihead? What is the need for so many parameters?

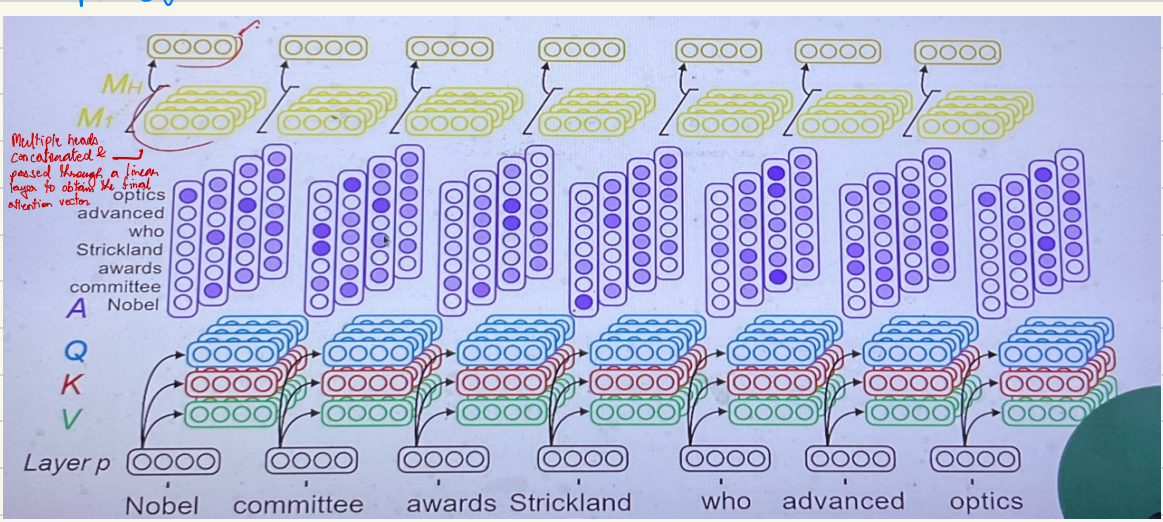
A) Idea is influenced by CNN. Just like in CNNs we apply multiple kernels and try to learn different aspects of an image (eg: ear, eye, nose, etc. in faces. Each kernel identifies with a different aspect of the image). Similarly, with multiple heads, the hope is to learn different kinds

of information about a token, (eg: is it, a subject?, is it a noun or pronoun?, Is it a named entity?, etc.)

NOTE: We don't know which attention head looks at which aspect. This is just for illustration - to show the intent of using multiple heads

*) Now, for every token, we have multiple attention vectors - each one of dimension 'd' i.e the dimension of the input vector (or input embedding). So, in the above example, we have 3 attention vectors of dimension 'd'. To use them we concatenate all the vectors, and now we have a vector of dimension '3d'

*) However, the next layer is also a self-attention layer (stacked on top of this one) with input dimension as 'd'. So, to convert our '3d' dimension vector to a 'd' dimension vector, we pass it through a linear layer of size '3d x d'



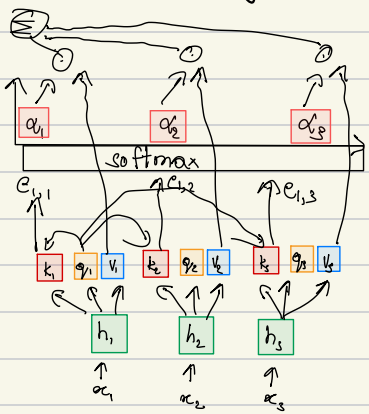
Q) What is the complexity of self-attention block with 'N' tokens?

A) $O(N^2)$ → each token attends to every other token

* In feed forward networks, data flows forward & in that direction only with no loops or cycles (unlike RNNs where data is repeatedly fed across different timesteps)

Adding Non-Linearities

Problem: Self-attention layer is a linear transformation of the previous layer with non-linear weights



$k_t = W_k h_t \qquad q_t = W_q h_t \qquad v_t = W_v h_t$

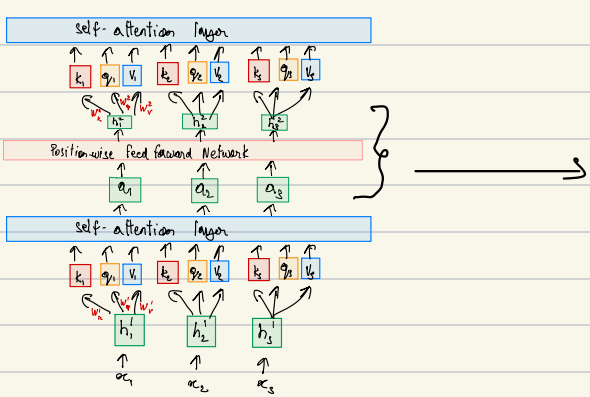
$\alpha_{i,t} = \exp(e_{i,t}) / \sum_{t'} \exp(e_{i,t'}) \qquad [e_{i,t} = q_i \cdot k_t]$

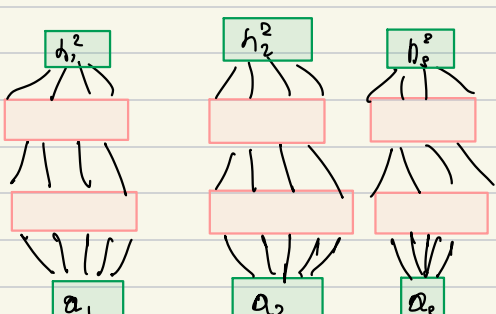
$a_i = \sum_t \alpha_{i,t} \cdot v_t = \sum_t \alpha_{i,t} W_v h_t = W_v \sum_t \alpha_{i,t} \cdot h_t$

W_v: linear transformation; $\sum \alpha_{i,t} \cdot h_t$: non-linear weight

* Linear models have many problems as they're unable to model complex distributions.
* To tackle this issue, we pass the output through another layer called the - 'feed forward layer'* (similar to a simple Multi-Layer Perceptron)

Q) So how is applied here?





↳ Remember: These are vectors

- We take each attention vector and apply a feed-forward network at each position i.e., individually to each attention vector (as shown above)
- It consists of two linear transformations with a non-linear activation in between (ReLU was used in transformer but we can try other things like sigmoid)

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

($\max(0, \cdot)$ → ReLU; x → attention vector)

* Introduces 2 new set of weights W_1 & W_2 (ignoring bias) W_1 & W_2 are shared across all tokens.
* Called position-wise feed forward network because the FFN is applied at each position (or at each attention vector)

Q) what is the purpose of this?
A) Self-attentⁿ allows us to exchange informatⁿ with different tokens (like message passing in GNNs). This can also be seen as some sort of memory fetching i.e., fetching key from each token as per the query and doing some processing. The actual processing after that happens in the FFN.

Self attention → fetching/message passing
FFN → Processing after exchange of message

FFN is also considered as the internal memory of the Transformer model. This is because the FFN has a huge no of parameters - eg: if the embeddings are of size 512, then W_1 will be of size $512 \times A$. Also, generally we first up-project the vector to a higher dimension followed by down-projection instead of

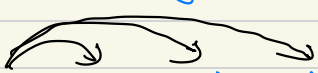
the other way around to avoid losing information. So, A will be greater than 512 which means huge no of parameters.

*) Size of the hidden state is a hyperparameter.

*) Input, self-attention & feed forward are the basic units of a transformer block in the transformer model. This block is repeated multiple times (6 in base transformer model & 12 in XL model)

Masked Decoding

We've missed out a very subtle detail in self-attention. We know that each query attends to every key in the sequence.



eg: I am going to school

When considering the decoder (in a machine translation task), its task is to generate the sequence of outputs. So if we consider a sentence 'मैं स्कूल जा रहा हूँ' (Translation: I am going to school) & when we generate 'I' the corresponding decoder doesn't know what must come in the next state (or timestep) [Because till now we've only generated till 'I']

So, 'I' has no knowledge of tokens in the 2nd, 3rd or any position after it (in fact it wouldn't even know how many positions will come after it). So, the decoder does not have access to any future states whereas in the encoder (which takes in the hindi sentence), each word has access to every other word.

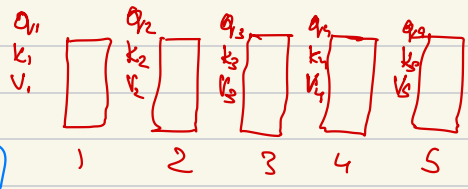
So, during inference, we generate 1 token, then based on that

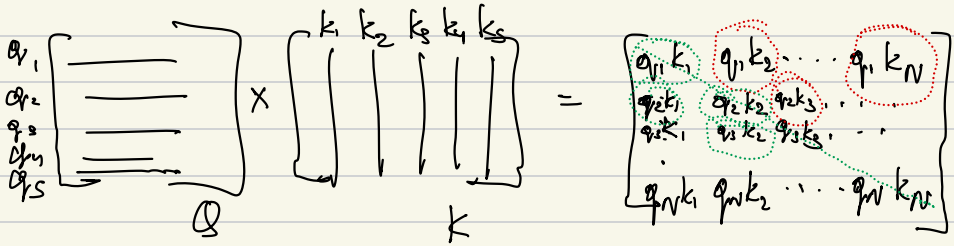
we generate the next and so on. Normal self-attention is not applicable here as each token only has access to the tokens generated before it to perform self-attn.

During training, we have both the source & target sentences. If we access the future decoder states during training then we are essentially just 'cheating' because during inference we won't have that information. So, we need to do some tweaks such that during training as well, no decoder state has access to future tokens.

Q) How to do this?

- Lets say we have q, k, v vectors obtained from each decoder state
- We need to ensure that 5 only has access to {4, 3, 2, 1}, 4 only has access to {3, 2, 1} and so on.
- In practice, we are not actually doing vector operations but actually matrix operations (to leverage the parallel processing powers of devices like GPUs & TPUs)

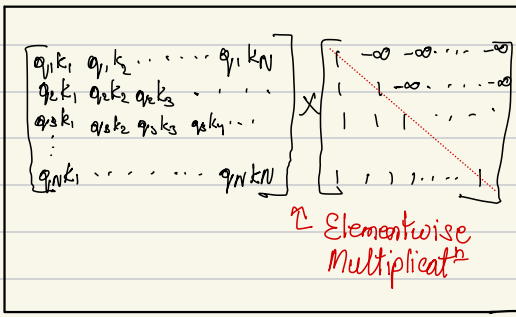




- As per our requirement we need to ensure that q_1 cannot access k_2, k_3, k_4 and soon. In general, any q_i should not be able to access any k_j such that $j > i$ i.e., the upper half of the resultant matrix should not be accessible

i.e., the part above the diagonal.

- To ensure this we do an element wise multiplicatⁿ where all the elements above the primary diagonal are $-\infty$.



↳ Why?

↳ Because these elements are query-key dot products and this operatⁿ is followed by a softmax → $e^x / \sum e^x$

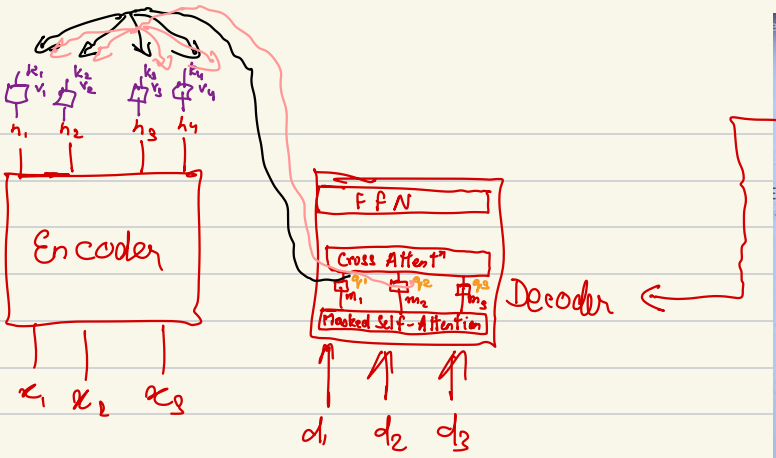
↳ Multiplying the values with $-\infty$ sets them to $-\infty$ which ensures that after softmax these weights are converted to 0 ($\because e^{-\infty} = 0$) & the rest of the weights are considered as it is.

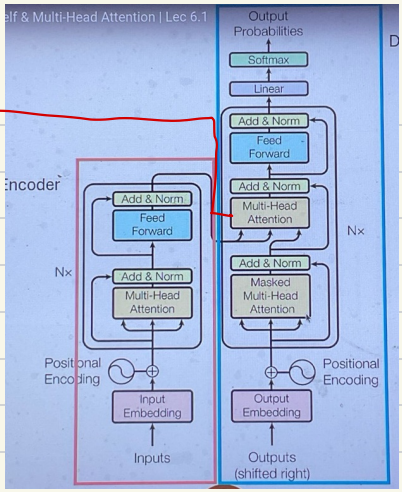
- Thus, in this way we prevent any token from accessing any information from future tokens during training.

During test time inferencing, we have already learnt W_Q, W_K & W_V matrices. After generating the 1st token ('I' in the previous example), we pass it back as input again to the decoder to generate the next token and keep repeating until the <EOS> (end of sequence token) is output by the model.

Q) The final layer of the transformer encoder will also output a set of attention vectors. But we have not established any connection b/w the encoder & decoder networks. So, how is the output of the encoder linked to the decoder?

A) We do this through another layer known as cross-attention layer





- In this cross attention, after we get the output of the masked self-attention, we pass it through a linear layer to obtain the query vectors. Then we generate the key & value vectors from the output of the final encoder layer. Each decoder query attends to keys from the encoder outputs and the remaining process is the same as in self-attention.

NOTE: There is no concept of masking in cross attention because key & value vectors are generated from the input & decoder states (which generates query vectors) will attend to these keys.

Lec 6.2 - Intro to Transformers: Positional Encoding & Layer Normalization

Survey Paper: Position Information in Transformers: An Overview
- Philip Dufter (arXiv)

Positional Encodings