Week 2

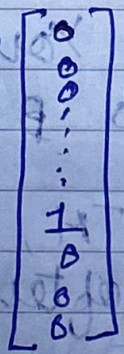
* Word Embeddings :

Word Representation

V = [a, aaron, ... , zulu, <UNK>]

1 hot representation :

- Man (5391) : O_{5391}



↳ O of one-hot

Disadvantage: Does not generalise well
Eg: I want orange juice, it cannot guess I want apple juice here.

* Featurized Representation: Word Embeddings

	(5391) Man	(9853) Woman	(4914) King	(7157) Queen	(456) Apple	(6257) Orange
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
⋮	⋮	⋮				
Size						
Cost						
Verb						

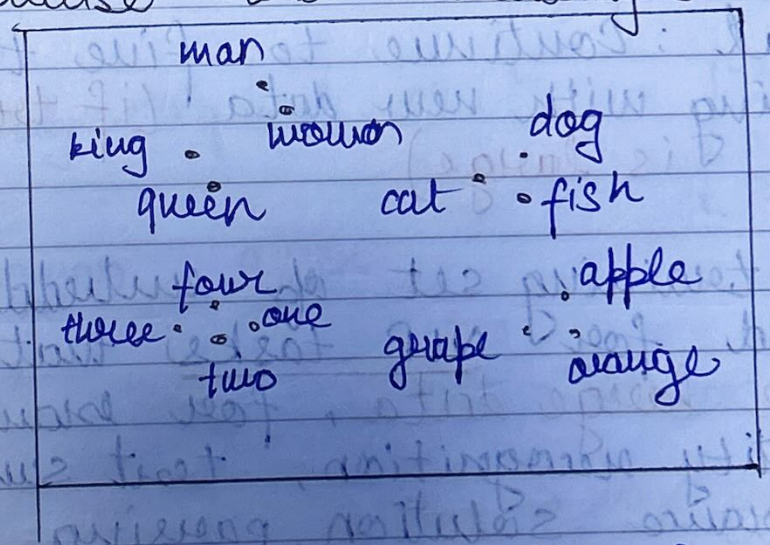
300

e_{9853}

300 dimensional vector to represent man, known as EMBEDDING

e_{5391} ⇒ notation

- Visualise Word Embeddings



done using t-SNE

Name Embedding Entity Recognition

Sally Johnson is an orange farmer
Robert Linn is an apple farmer
a durian cultivator

given orange farmer, can deduce apple farmer.

- Using the features & similarity in word embeddings can also deduce that durian & cultivator are similar to fruits & farmer respectively.

- We also use transfer learning to then include the new deduced words from here as part of vocabulary for other examples.

* Transfer Learning & Word Embeddings:

A → B

1. Learn word embeddings from large text corpus (1-100 B words)
   → Or download pre-trained embedding online
2. Transfer embedding to new task with smaller training set (say, 100K words)
3. Optional: Continue to fine tune the embedding with new data (if training set in B is large)

- This training set of embeddings can be used for NLP tasks that do not require large data, for example:
  ** Named entity recognition, text summarization, co-reference solution, parsing

- Implementing transfer learning on set 1, the advantage we get is to use less training data, smaller labelled text *

We can use relatively lower dimensional feature vectors. Thus instead of 10,000 dimensional one-hot, use 300 dimensional dense vector.

- Transfer learning is not useful for language modelling and ML settings.
- Word embeddings show similarity to face encoding. Except here we may have a fixed vocab & there an unknown sea of images.

* Properties of Word Embeddings:

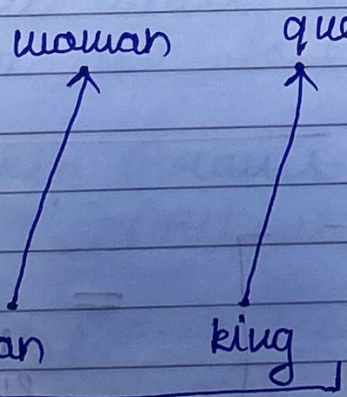
It can develop analogies:

Eg: man → woman then king → ?

(take ref. from table 1 in word embedding)

$e_{man} - e_{woman} \simeq \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

$e_{king} - e_{queen} \simeq \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$



easier version of t-SNE (300D → 2D) for understanding, not common to have such easy llgm like relation

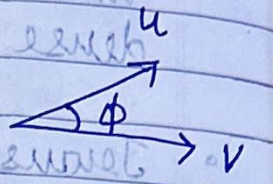
$e_{man} - e_{woman} \approx e_{king} - e_w$

$e_w \Rightarrow \max_{\text{maximum}} \underbrace{\text{sim}}_{\text{similarity func}^n}(e_w, e_{king} - e_{man} + e_{woman})$

* Similarity func^n:

COSINE SIMILARITY

$$sim(u, v) = \frac{u^T v}{||u||_2 ||v||}$$

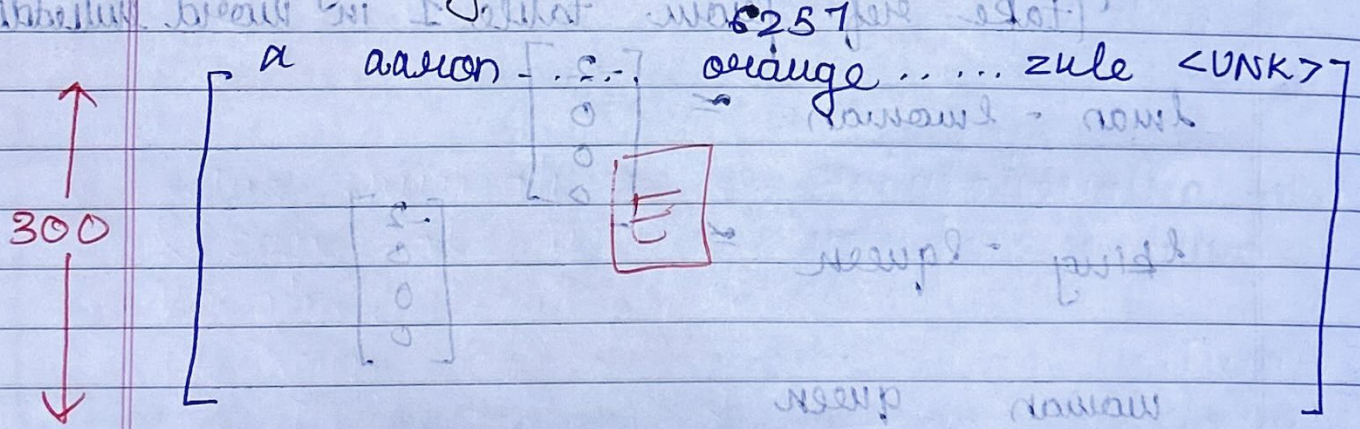


→ called cosine similarity as the formula is the cosine of u & v.

$||u - v||^2$ ⇒ Euclidean distance

⇒ rather a measure of dissimilarity

* Embedding Matrix



$E \cdot O_{6257} = e_{6257}$

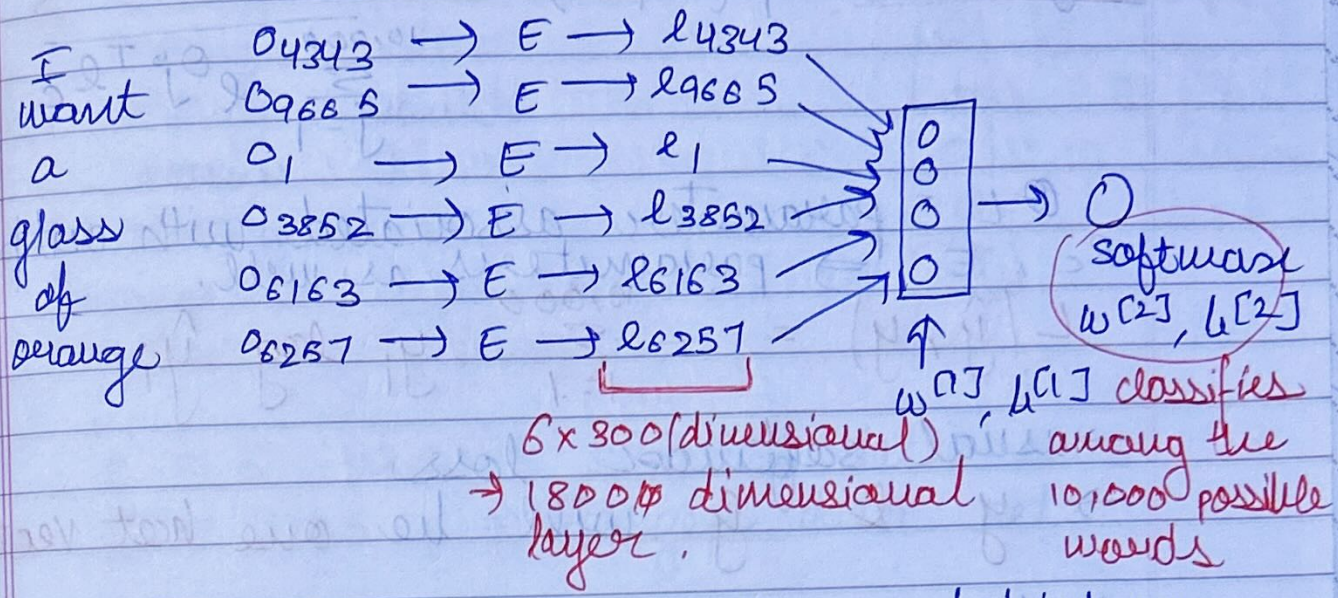
(300, 10k) (10k, 1) (300, 1)

embedding vector

E = Embedding matrix O = One hot matrix

* Learning Word Embeddings :

I want a glass of orange juice.



if we always use a 4-word history, or a fixed history, then the dimensions of the layer will be (4×300 =) 1200 & thus it allows us to compute very long sequences as well.

If you want to learn a language model, you use the last 4 words. However if you want to learn word embeddings then you use last 1 word or nearby 1 word.

* Skip Grams (Word2Vec)

Context → Target pairs

Model :

Vocab size = 10,000

Mapping : context c ("orange") → target t ("juice")

6257 4834

$$e_c = E\, o_c$$

$o_c \rightarrow E \rightarrow e_c \rightarrow O \xrightarrow[\text{softmax}]{} \hat{y}$

↳ key

Softmax: $$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

θ_t = parameter associated with output t

e_c, E ⇒ parameters as well.

$$L(\hat{y}, y) = -\sum_{i=1}^{10,000} y_i \log \hat{y}_i$$

→ usual softmax loss

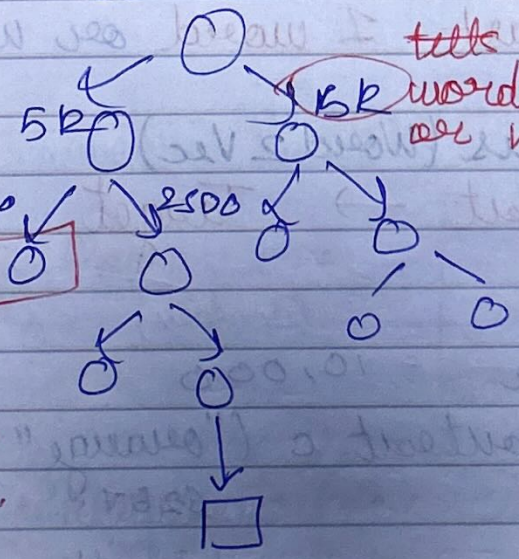
→ y & $\hat{y}$ will be one hot vectors

* Problems with Softmax:

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

this summation is computationally expensive & can take very long for bigger vocabs.

We can use a **Hierarchical Softmax**:



tells classifies if our word is in first 500 or not

& so on down the tree

Each node is a Binary classifier (Logistic).

Read word2vec original documentation
→ Thomas Mikolov, Sasaki, Chen, Greg, Jeff Dean

* Negative Sampling :

x		y
context	word	target ?
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0
(orange)	(juice/king/etc)	(1/0)

k (-ve samples)

→ context → target → label

k = 5 - 20 ⇒ smaller data-sets
k = 2 - 5 ⇒ larger data sets.

So, the problem statement is:
Given a pair of words, like orange & juice, do you think they appear together? Or we got them by sampling 2 words close to each other or as one word from text & the other chosen at random.

Model:

$$P(y=1 \mid c,t) = \sigma(\theta_t^T e_c)$$

think as k:1, negative to positive

$O_{6257} \rightarrow E \rightarrow e_c$

10,000 binary classification problems

→ on every iteration, we use only (k+1) words, where 1 word is the answer & there are k randomly chose -ve values.

the k will be fixed & given?

How to choose -ve samples?

1. Choose according to empirical importance, but that may words like: the, and, a, for, etc gain a lot of importance.
2. Other is to divide by $1/V$, sampling -ve examples uniformly. '$1/V$'

very non-representative.

Thus middle ground is :

$$P(w_i) = \frac{(f(w_i))^{3/4}}{\sum_{j=1}^{10,000} (f(w_j))^{3/4}}$$

Thus softmax vs -ve sampling
→ summation over 1000 binary classification.

* GloVe (global vectors for word representation)
→ count that captures how often they appear together
X_{ij} ⇒ times j appears in context of i
X_{ij}: i = c, j = t
j = t (target), i = c (context)

possible to have $X_{ij} = X_{ji}$
choice of context & target ⇒ symmetric
words appear in +/- 10 words of each other.
→ not symmetric for choice of context words being immediately before/after.

Model:

$$\text{minimise} : \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(x_{ij}) \left(\theta_i^T e_j + b_i + b'_j - \log x_{ij}\right)^2$$

weighted terms,

$f(x_{ij})$
- → to give $f(x_{ij}) = 0$ for $x_{ij} = 0$
- → does not give too much importance to: a, and, the, etc.
- → does not give too little importance to: durian, etc.
- → thus valid only when the words have co-occured atleast once.

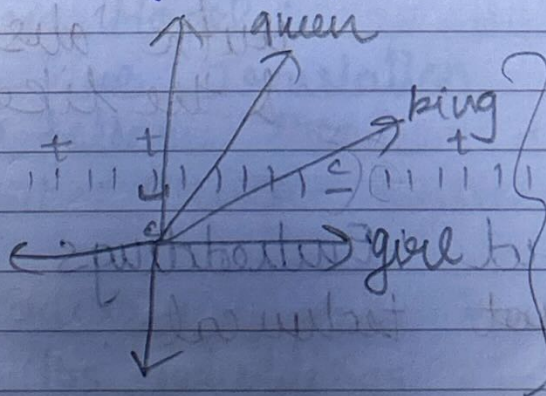
→ here, θ_i & e_j are symmetrical, as they play pretty much the same role & we could reverse them or sort them & achieve same objective.

$$\text{~~final~~} = e_w^{\text{final}} = e_w + \theta_w.$$

With an algo like this, we cannot guarantee direction of axes & minimise priority of features as:

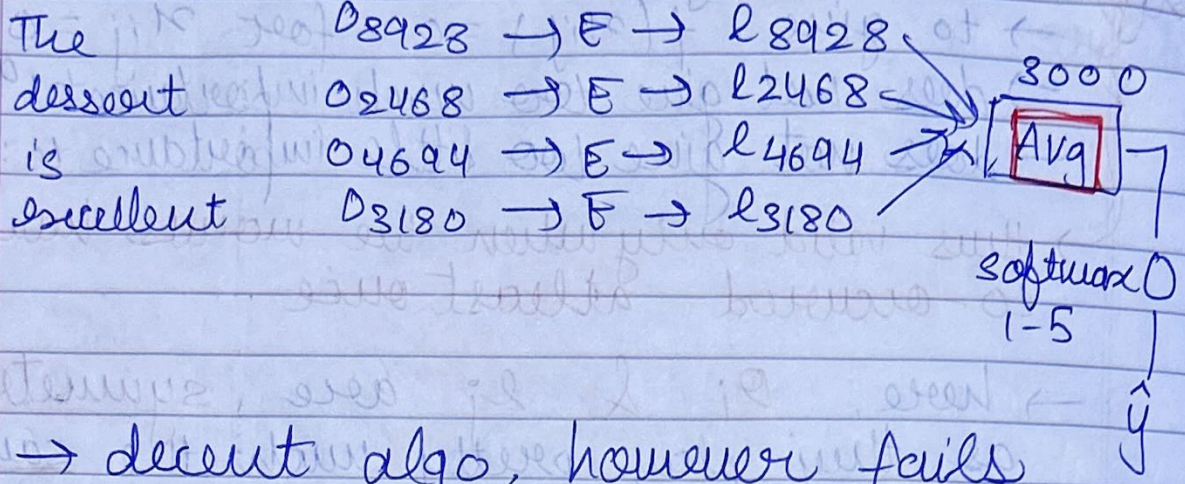
together

$$\theta_i^T e_j \Rightarrow (A\theta_i)^T (A^{-T} e_j) = \theta_i^T A^T A^{-T} e_j = \theta_i e_j$$



* Sentiment Classification :

The dessert is excellent ★★★★★
8928 2468 4694 3180



→ decent algo, however fails when:
The hotel lacked good ambience, good taste and good service.
→ Acc to avg, good will give this 3/5 star while it is 1 star.

→ Use RNNs with word embeddings
to deal with avg's disadv. & consider lacked
to deal with sequences that replace 'lacked' with 'absent' & the like.

* Debiasing Word Embeddings

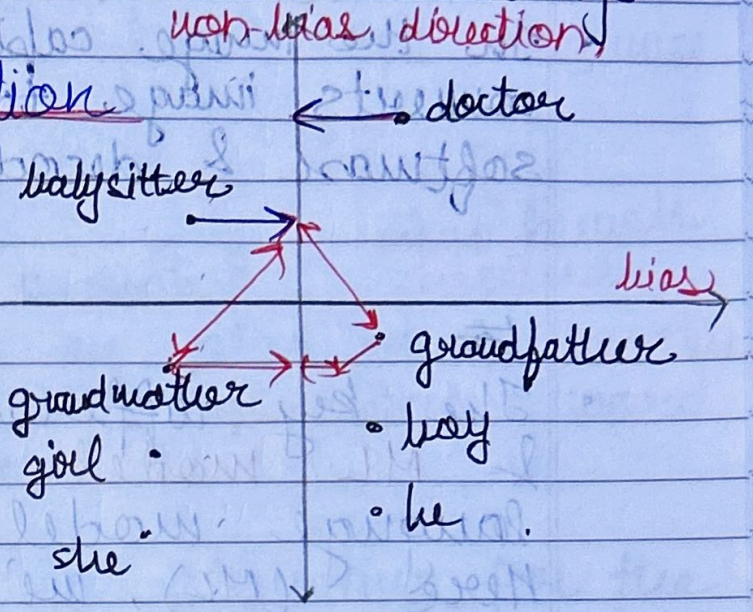
literature bias, not technical.

man : woman as king : queen
man : computer_programmer as woman : homemaker ✗

→ Father : Doctor as Mother : Nurse ✗
reflect biases of the text used to train ⇒ word embeddings.

* Addressing bias in word embeddings

1. Identify bias direction.
   - he - she
   - male - female
   - ⋮
   - $ average



2. Neutralise :
For every word that is [not definitional], project to get rid of bias.
→ not affected by bias.

3. Equalise pairs.