☆ LSTM (Long Short term memory) Unit.

- first differece b/w equation b/w GRU And LSTM.

⇒ GRU

$\tilde{c}^{\langle t \rangle} = \tanh(W_c[\Gamma_r * c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c)$ — $a^{\langle t-1 \rangle}$

$\Gamma_u = \sigma(W_u[c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u)$

$\Gamma_r = \sigma(W_r[c^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_r)$ } 2 gates in GRU

LSTM has 3 Gates.

$c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + (1 - \Gamma_u) * c^{\langle t-1 \rangle}$

Γ_f

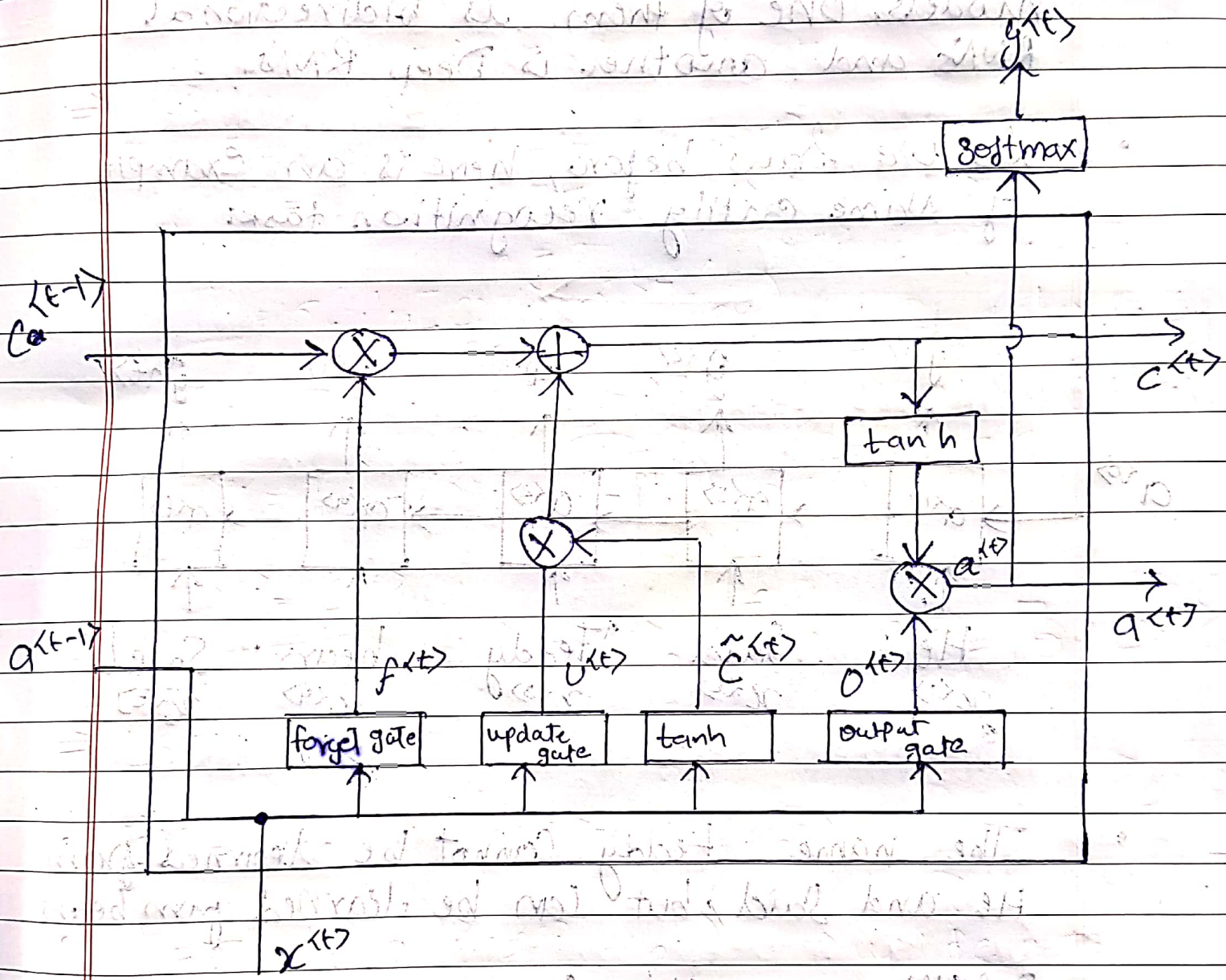
$a^{\langle t \rangle} = c^{\langle t \rangle}$

$a^{\langle t \rangle} != c^{\langle t \rangle}$, Instead $a^{\langle t \rangle} = \Gamma_o * c^{\langle t \rangle}$

⇒ LSTM

- $\tilde{c}^{\langle t \rangle} = \tanh(W_c[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_c)$
- $\Gamma_u = \sigma(W_u[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_u)$ → update gate
- $\Gamma_f = \sigma(W_f[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_f)$ → forget gate
- $\Gamma_o = \sigma(W_o[a^{\langle t-1 \rangle}, x^{\langle t \rangle}] + b_o)$ → output gate
- $c^{\langle t \rangle} = \Gamma_u * \tilde{c}^{\langle t \rangle} + \Gamma_f * c^{\langle t-1 \rangle}$
- $a^{\langle t \rangle} = \Gamma_o * \tanh c^{\langle t \rangle}$

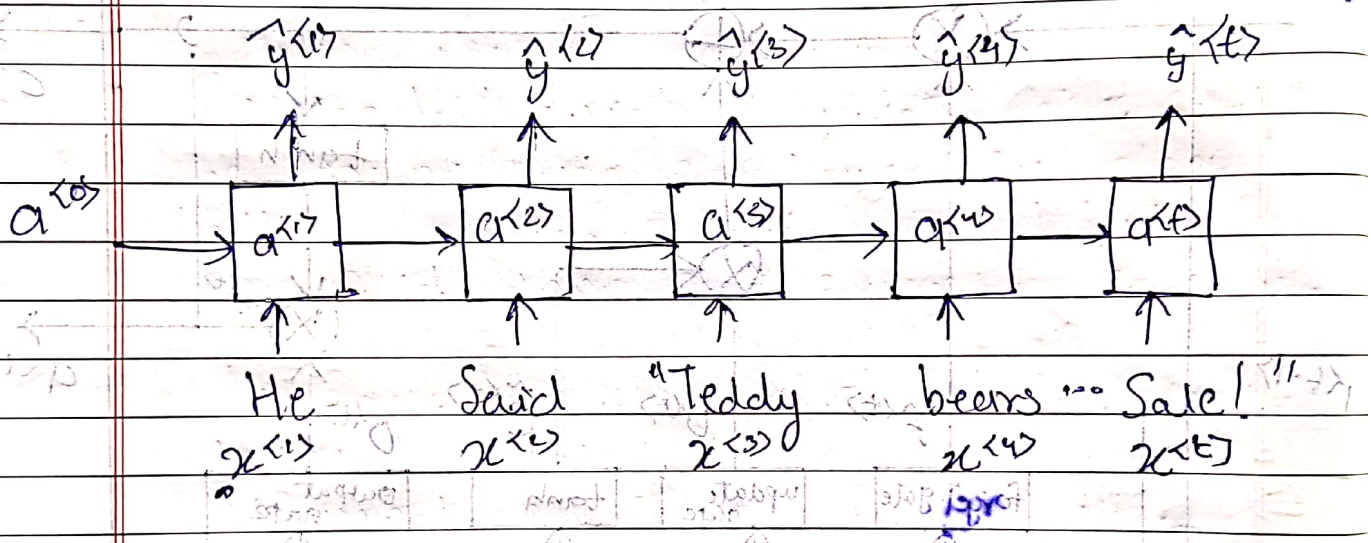
- LSTM - the other type of RNN that can enable you to account for long-term dependencies. It's more powerful and general than GRU.
- In LSTM, $c^{\langle t \rangle} != a^{\langle t \rangle}$
- In GRU we have update gate $U (\Gamma_u)$, a relevance gate $r (\Gamma_r)$, and a candidate cell variables $\tilde{c}^{\langle t \rangle}$ while in LSTM we have an update gate U (Sometime its called input gate I), a forget gate F, and output gate O, and a candidate cell variable $\tilde{c}^{\langle t \rangle}$.
- Some variants on LSTM includes:
  - ↳ LSTM with peephole connection.
  - → The normal LSTM with $c^{\langle t-1 \rangle}$ include with every gate.
- There isn't a universal superior between LSTM and it's variants. One of the advantages of GRU is that it's simpler and can be used to build much bigger network but the LSTM is more powerful and general.



☆ Bidirectional RNN

- There are still some ideas to let you build much more powerful Sequence models. One of them is bidirectional RNNs and another is Deep RNNs.

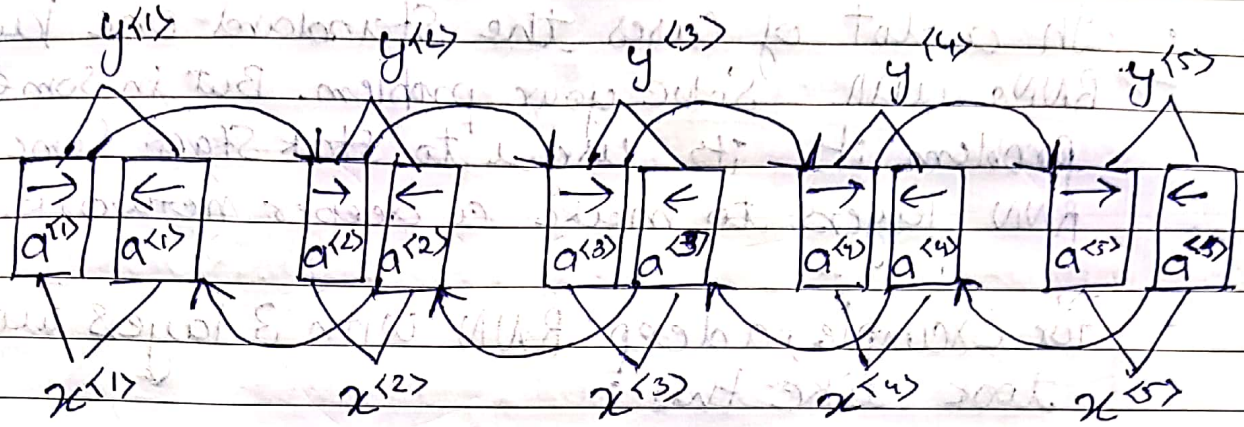
- As we saw before, here is an example of Name entity recognition task:



- The name teddy cannot be learned from He and Said, but can be learned from bears

- BiRNN fixes this issue.

- Here is BRNN architecture



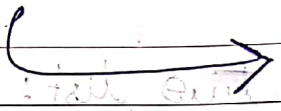
here output will be of two list: $\hat{y}=[\ldots]$, $\hat{y}_{rev}=[\ldots]$
OR Inshort we can right; $\hat{y}^{\langle t \rangle} = g(W_y[\vec{a}^{\langle t \rangle}, \overleftarrow{a}^{\langle t \rangle}] + b_y)$

- Note, that BRNN is an acydic graph
- part of the forward propagation goes from left to right and part from right to left. It learn from both sides.
- To make prediction we use $\hat{y}^{\langle t \rangle}$ by using the two activation that come from left and right.
- The blocks here can be any RNN block including the basic RNN, LSTM or GRU.
- for a lot of NLP or text processing problems, a BRNN with LSTM appears to be commonly used.
- The disadvantage of BRNN that you need the entire sequence before you can process it. for example, in live speech recognition if you use BRNN you will need to wait for the person who speaks to stop to take the entire sequence and then make prediction.

*enlivo
Date : ___/___/___
Page : ___

★ Deep RNN

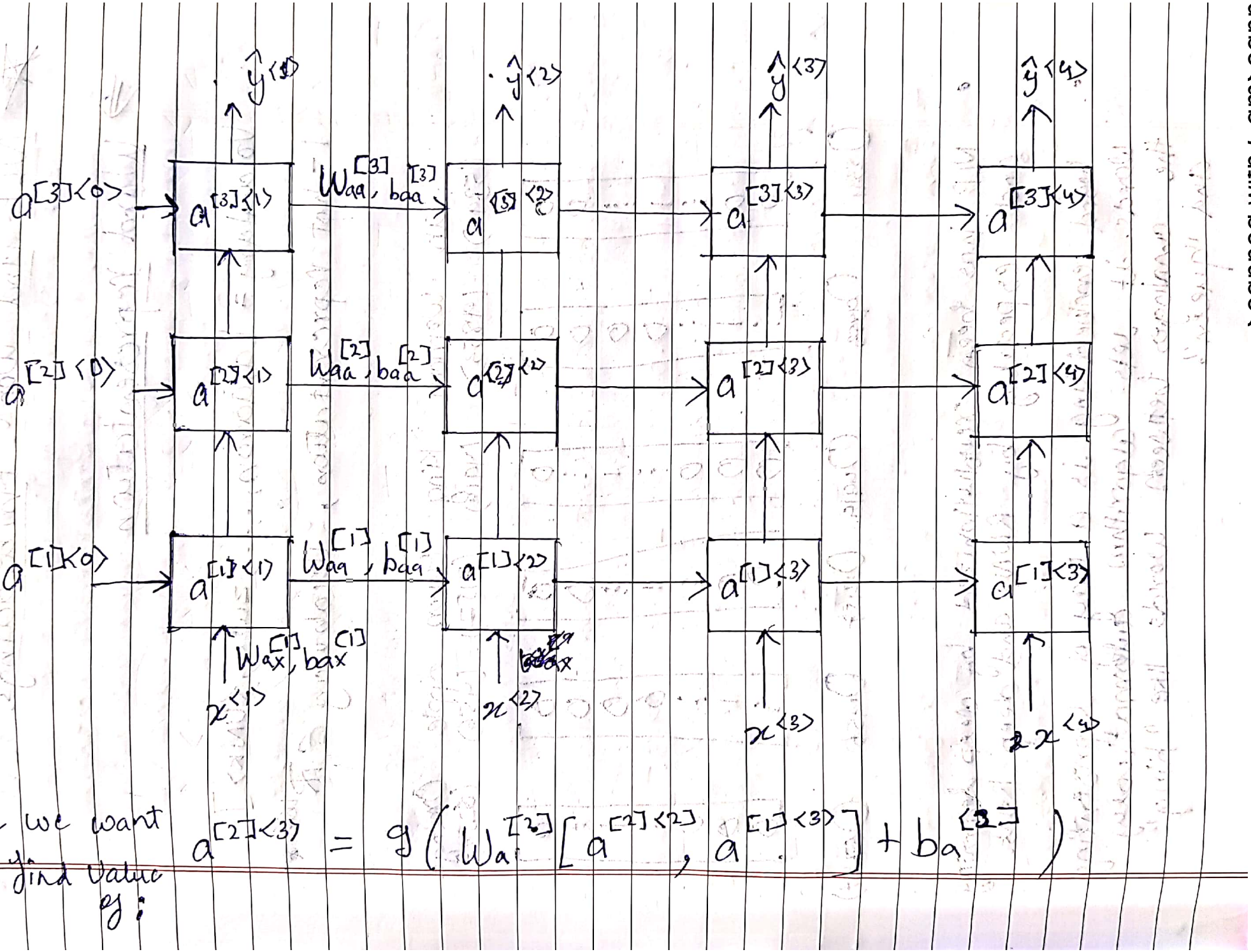
- In a lot of cases the Standard one layer RNNs will Solve your problem. But In Some problem its its Useful to ~~Stak~~ Stack Some RNN layers to make a deeper network.
- For Example, a deep RNN with 3 layers would look like this:



- In feed-forward deep nets, there could be 100 or even 200 layers, In deep RNNs Stacking 3 layers is already Considered deep and expensive to train
- In Some cases you might See Some feed-forward network layers Connected after recurrent cell, Just like in CNN.

⇒ Back propogation with Deep RNN.

- In modern deep learning frameworks, you only have to implement the forward pass and the framework takes care of the backward pass, So most deep learning Engineers do not need to bother with the details of the backward pass. If however you are an expert in ~~calculus~~ and want to see



Suppose we want find value of:

$$a^{[2]\langle 3\rangle} = g\left(W_a^{[2]}\left[a^{[2]\langle 2\rangle}, a^{[1]\langle 3\rangle}\right] + b_a^{[2]}\right)$$

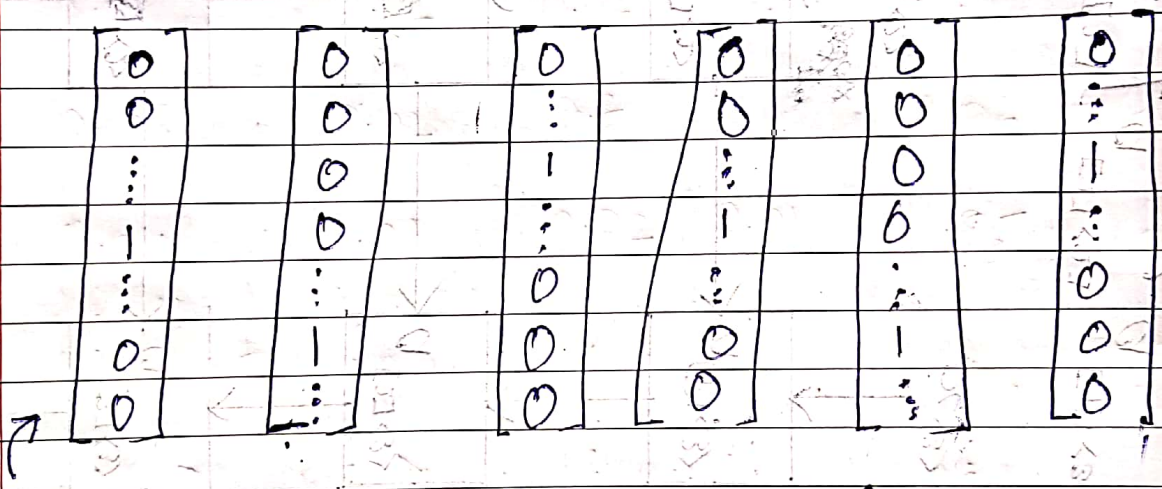
★ NLP and Word Embeddings.

⇒ Word representation

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

1-hot representation, Vocabulary $\in \mathbb{R}^{10000}$

Man	Woman	King	Queen	Apple	Orange
(5391)	(9853)	(4914)	(7157)	(456)	(6257)



O_{5391} O_{9853} O_{4914} O_{7157} O_{456} O_{6257}

- NLP has been revolutionized by deep learning and especially by RNNs and deep RNNs.
- Word embedding is a way of representing words. it lets algorithm automatically the analogies between words like "king" and "queen".

- So far we have defined our language by a vocabulary. Then represent our words with a one-hot vector that represents the words in the vocabulary. As shown on previous page

- We will use the annotation O_index for any word that is represented with one-hot like in the image.

- One of the weakness of this representation is that it treats a word as a thing that itself and it doesn't allow an algorithm to generalize across words.

  ↳ For example: "I want a glass of ~~apple~~ orange ____". , a model should predict the next word as juice.

  ↳ Similar example: "I want a glass of appk ____?" , a model won't easily predict juice here if it wasn't trained on it. And if so the two example aren't related although orange and apple are similar.

- Inner product between any one-hot encoding vectors is zero. Also, the distances between them are same.

- So, ~~Intead~~ Instead of a one-hot encoding ~~tec~~ presentation, won't it be nice if we can learn a featurized representation with each of these words: man, woman, king, queen, apple and orange?

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
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
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Verb	⋮	⋮	⋮	⋮	⋮	⋮
	e_{5391}	e_{9853}	e_{4914}	e_{7157}	e_{456}	e_{6257}

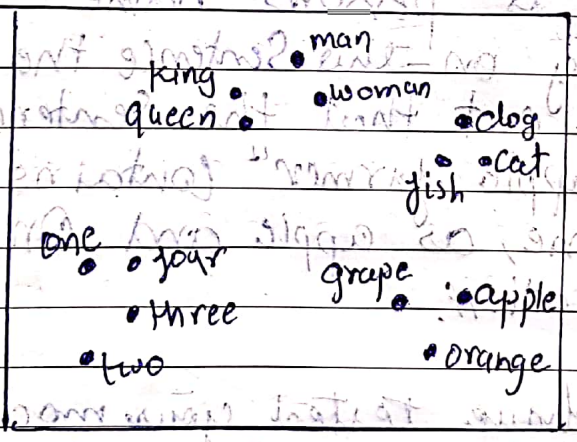
300

→ Each word will have a, for example, 300 feature with a type of float point number,

- Each word column will be a 300-dimensional vector which will be the representation

- We will use the notation e_{5391} to describe 'man' word features vector.
- Now, if we return to the example we described again:
  - ↳ "I want a glass of orange."
  - ↳ "I want a glass of apple."
- Orange and apple now share a lot of similar features which makes it easier for an algorithm to generalize between them.
- We call this representation 'Word Embedding'.

⇒ To visualize word embedding we use a t-SNE algorithm to reduce the feature to 2 dimension which makes it easy to visualize

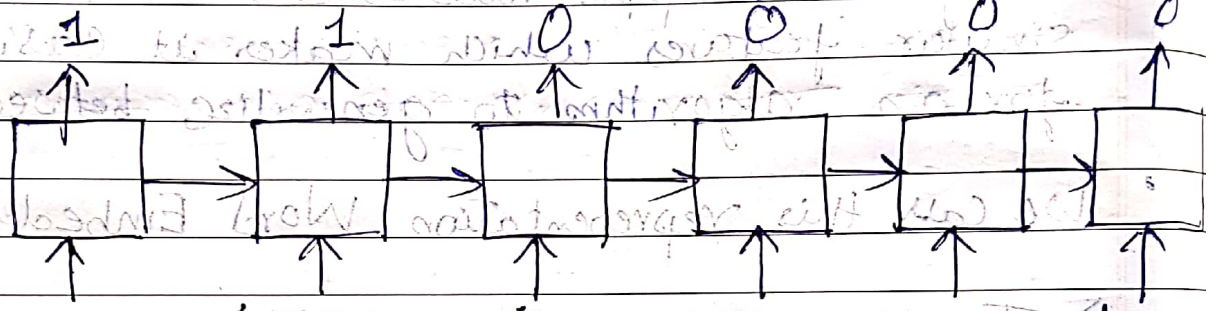


- You will get sense that more related words are closer to each other.

→ The Word-Embedding came from that we need to embed a unique vectore inside a n-dim space.

☆ Using Word Embeddings.

- Let's see how we can take the feature representation we have extracted from each word and apply it in the named entity recognition problem.
- Given this example (from named entity recognition)



Sally Johnson is an orange farmer

- Sally Johnson is person's name
- After training on this sentence the model should find out that the sentence "Robert Lin is an apple farmer" contains Robert Lin as a name, as apple and Orange have near representations.
- Now if you have tested your model with this sentence "Mahmoud Badry is a durian cultivator" the network should learn the name even if it hasn't seen the word durian before (during training). That's the power of word representation.

- The algorithm that are used to learn word embedding can examine billions of word of unlabeled text - for example, 100 billion words and learn the representation from them.

- Transfer learning and word embeddings:

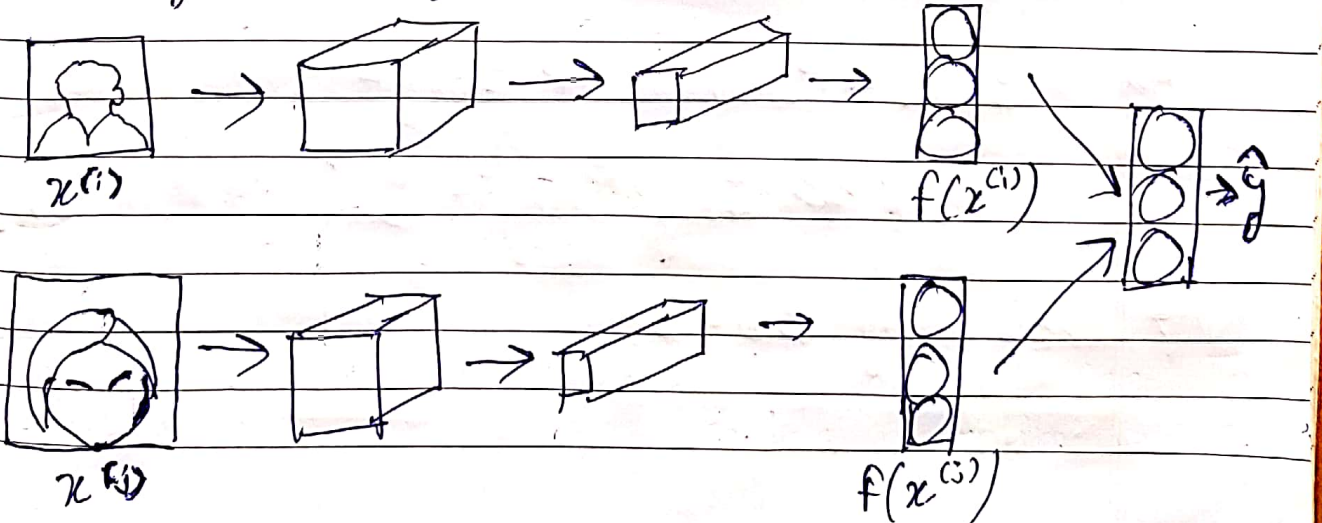
i) learn word embeddings from large text corpus. (1-100 Billion of words)
   - or download pre-trained embeding online.

ii) Transfer embedding to new task with the smaller set (say, 100k words)

- Word embedding tend to make the biggest difference when the you're trying to carry out has a relatively small training set.

- Also, one of the advantages of using word embedding is that it reduce the size of the input:
  ↳ 10,000 one hot compared to 300 feature vector.

- Work embedding have an interesting relation to the face recognition task:



- In this problem, we encode each face into a vector and then check how similar are these vector.
- words encoding and embeddings have similar meaning here

→ In the word embedding task, we learning a representation for each word in our vocabulary (unlike in image encoding where we have to map each new image to some n-dimensional vector). ~~we discuss the algorithms in next section.~~

☆ Properties of word embedding.

- One of most fascinating properties of word embedding is that can also help with analogy reasoning. While analogy reasoning ~~many~~ may not be by itself the most important NLP application, but it might help convey a sense of what these word embedding can do.

- Analogies Example:
  → Given this word embedding table:

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
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	0.70	0.69	0.03	-0.02
food	0.09	0.01	0.02	0.01	0.95	0.97
	e_{man}	e_{woman}	e_{king}	e_{queen}	e_{Apple}	e_{orange}

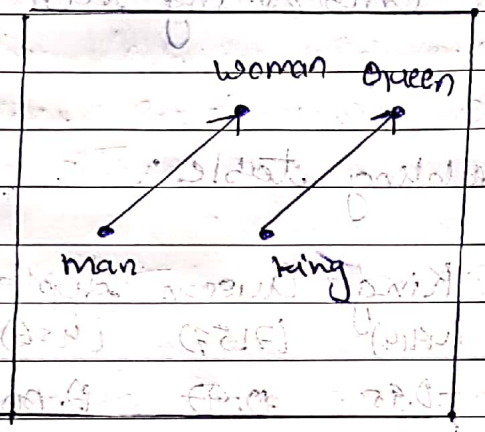
- Can we conclude this relation:
  ↳ Man → Woman
  ↳ King → ??

- Lets substract e_{man} from e_{woman}.

This will equal to Vector:

$C_{man} - C_{woman} = [-2\ 000]$

- Similar $C_{king} - C_{queen} = [-2\ 000]$
- So the different is about the gender in both



→ This vector represent the gender

→ The drawing visualization of the 4D vector in 2D form that has been extracted by a t-SNE algorithm. It's a drawing just for visualization. Don't relp on the t-SNE algorithm for finding ~~parallel~~ parallels.

- So we can reformulate the problem to find:

→ $C_{man} - C_{woman} \approx C_{king} - C_{??}$

$C_{??} \approx C_{king} - C_{man} + C_{woman}$

→ lets say $C_W = C_{??}$

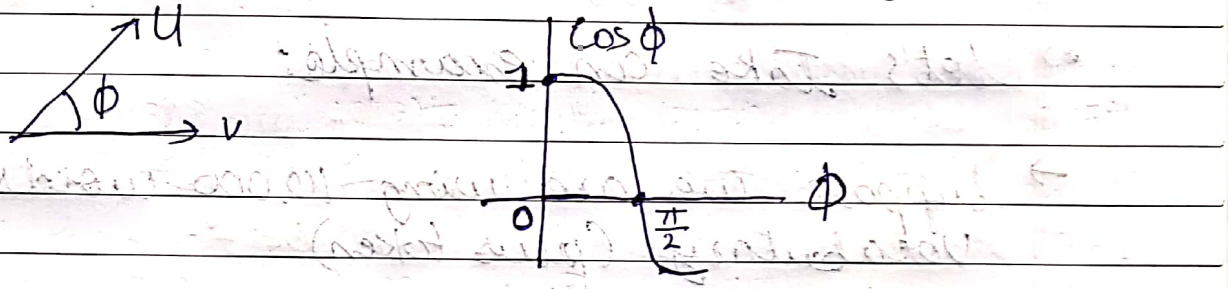
*enlivo
Date : ___/___/_____
Page: _____

∴ it can also be represented mathematically by:

→ $\text{argmax}\ Sim(C_w, C_{king} - C_{man} + C_{woman})$

⇒ $Sim(U,V) = \frac{U^T V}{\|U\|_2 \|V\|_2} = \frac{U^T V}{\sqrt{\sum_{i=1}^{k} U_i^2} * \sqrt{\sum_{i=1}^{k} V_i^2}}$

↳ this is called Cosine Similarity.



- Cosine Similarity - the most commonly used Similarity function.

→ In formula, The top part represent the inner product of U and V vector. It will be large if the vector are very similar.

- We can also use Euclidean distance as a Similarity function (but it rather measures a dissimilarity. So ~~you~~ we should take it with negative sign.)

- We can use this equation to calculate the Similarities between word embedding and on the analogy problem where $U = C_w$ and $V = C_{king} - C_{man} + C_{woman}$

☆ Embedding Matrix

⇒ ~~learning word embeddings~~

- ~~Lets start learning some algorithms that~~ can

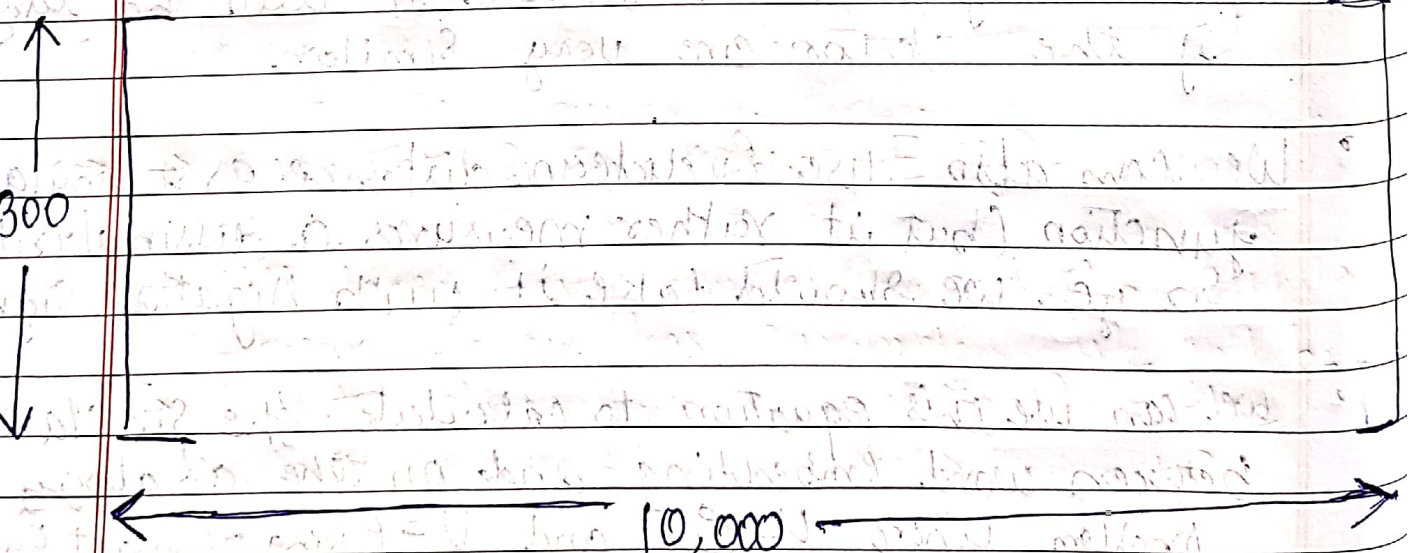
- when you implement an algorithm to learn a word embedding, what you end up learning is a embedding matrix.

- let's take an example:

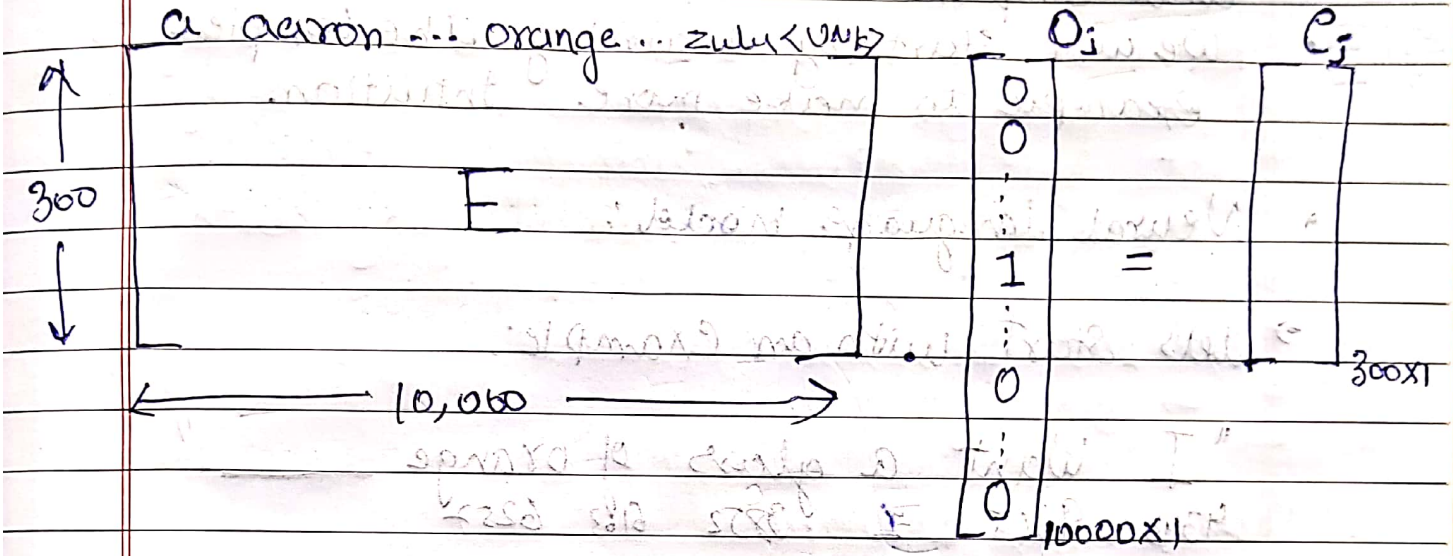
→ Suppose we are using 10,000 words as our vocabulary (plus token)

- The algorithm should create a matrix E of the shape (300, 10000) in case we are extracting 300 features.

a aaron .... orange .... zulu <UNK>



- If O_{6257} is the one hot encoding of the word Orange of shape (10000, 1), then, $np.dot(E, O_{6257}) = e_{6257}$ which shape is (300, 1)
- Generally $np.dot(E, O_j) = e_j$



- In the next sections, we will see that we first initialize E randomly and then try to learn all the parameters of this matrix.
- In practical it's not efficient to use a dot multiplication when you are trying to extract the embedding of a specific word, instead, we will use slicing to slice a specific column. In Keras there is an embedding layer that extracts this column with no multiplication.

★ Learning Word Embedding: Word2Vec & GLOVE

*enlivo
Date : __/__/____
Page: _____

⇒ learning word embeddings

- Let's start learning some algorithms that can learn word embeddings.
- At the start, word embeddings algorithms were complese but then they got simpler and simpler.
- We will start by learning the complex example to make more intuition.

- Neural language model:

→ Let's start with an example:

"I Want a glass of orange ____."
4343 9665 I 3852 6163 6257

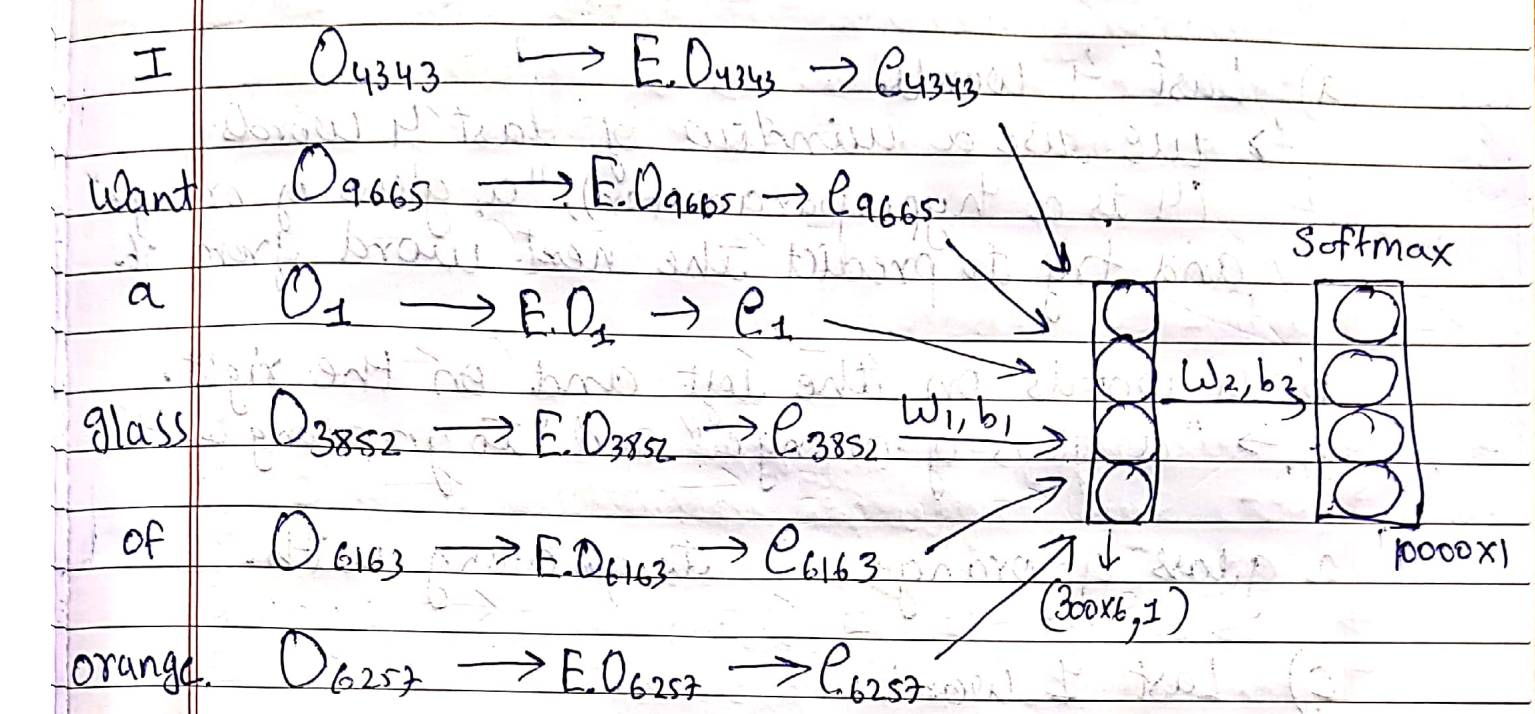
→ We want to build a language model so that we can predict the next word.

→ So we use this neural network to learn the language model

→ ↳ We get O_3 by np.dot(E, O_j)

↳ NN layer has parameter W_1 and b_2 while Softmax layer has parameters W_2 and b_2
↳ Input dimension is (300*6, 1) if the window size is 6 (six previous words)

One-hot encoding E.O;



→ Here $\hat{y}$ will be (10000x1) vector which represent any word from vocabulary, eg As y = 'Juice', it will backpropagate and learn all the weights according to true output.

→ We are optimizing E matrix and layer parameters. We need to maximize the likelihood to predict the next word given the context (previous words)

- This model was build in 2003 and works pretty decent for learning word E.
- In this ~~Exam~~ Example we took a window of 6 words that fall behind that we want to predict. There are other choices when we are trying to learn word embeddings.

↳ Suppose we have an example:

"I Want a glass of orange juice to go along with my cereal"

- To learn 'juice', choice of context are:

a) Last 4 words.
↳ we use a window of last 4 words (4 is a hyperparameter), "a glass of orange" and try to predict the next word from it.

b) 4 words on the left and on the right.
↳ "a glass of orange" and "to go along with"

a glass of orange ___ to go along with.

c) Last 1 word
↳ orange

d) Nearby 1 word.
↳ "glass" word is near juice.
↳ This is the idea of Skip grams model.
↳ The idea is much simpler and works remarkably well.
↳ we will talk about this in the next section.

- Researchers found that if you really want to build a language model, it's natural to use the last few words as a context. But if your main goal is really to learn a word embedding then you can use all of these other contexts and they will result in very meaningful word embeddings as well.

- To summarize, the language modeling problem poses a machines learning problem where you input the context (like the last four words) and predict some target words. And posing that problem allows you to learn good word embeddings.

☆ Word2Vec

- Before presenting Word2Vec, lets talk about SKIP-GRAMS.

↳ For example, we have the sentence:
"I want a glass of orange juice to go along with cereal"

↳ We will choose Context and target

↳ The target is choosen randomly ~~post~~ based on a window with a specific size.

Context	target	How far
Orange	juice	+1
Orange	glass	-2
Orange	my	+6

We have converted the problem into a supervised problem

↳ This is not an easy learning problem. learning within +10/-10 words (10 is an example) is hard. We want to learn this to get our word embeddings model.

- Word2Vec model.

→ Vocubulary size = 10,000 words
→ Let's say that the context word are 'c' and the Target word is 't'
→ We want to learn 'c' to 't'
→ We get e_c by $E.O_c$
→ We then use Softmax layer to get $P(t|c)$ which is $\hat{y}$
→ Also we will use the cross-entropy loss function.
→ This model is called Skip-grams model

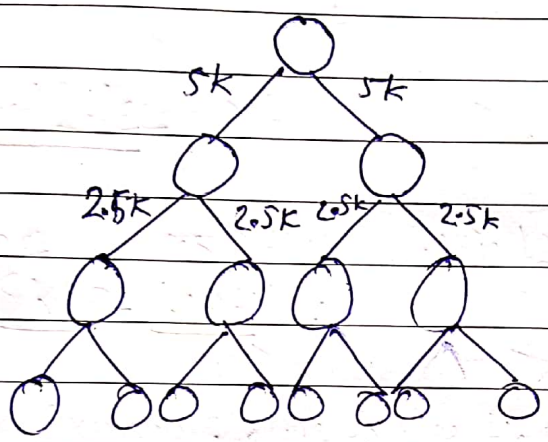
- The last model has a problem with the Softmax layer:

$$P(t|c) = \frac{\theta_t^T . e_c}{\sum_{j=1}^{10000} \theta_j^T . e_c}$$

→ Here we are summing 10000 number which corresponds to the number of words in our vocabulary.

→ if this number is larger say 1 million, the computation will become very slow.

- One of the solution for the last problem is to use "Hierarchical Softwax classifier" which works as a tree classifier.



- In practice, the hierarchical Softwax classifier doesn't use a balanced tree like the drawn one. Common words are at the top and less common are at the problem.

- How to Sample the context C?
→ One way is to choose the context by random from your corpus.
→ if you have done it that way, there will be frequent words like "the, of, a, and, to,..." that can dominate other words like "Orange, apple, durian..."
⇒ In practice, we don't take the context uniformly random instead there are some heuristics to balance the common words and the non-common words.

*enlivo
Date : ___/___/___
Page : ___

- Word2Vec paper includes 2 ideas of learning word embedding. One is Skip-gram model and another is CBoW (Continuous bag-of-words)