# <u>RNN</u>

⇒ Named entity recognition

X :- Harry ~~Potter~~ and Hermione Granger invented a new spell

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ . . . $x^{<t>}$ $x^{<9>}$

y :- 1 1 0 1 1 0 0 0 0

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$

$T_x = 9$
$T_y = 9$

⇒ $T_x^{(i)}$ } ith training sample
$T_y^{(i)}$

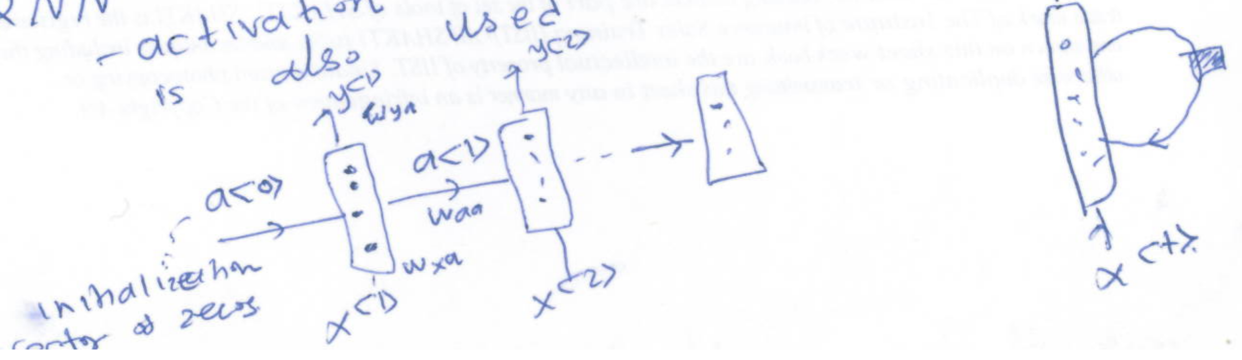
⇒ <u>NLP</u>

- dictionary of size 10k
- Now a days 30k to 100k are used often as well
- one-hot vector for each word
- <unk> token for unknown word.

⇒ <u>Why not</u> st<u>andard</u> <u>network</u> ?

- Problems:-
  - different length
  - Doesn't share features learned across different position of text

- RNN :- activation values from previous layer is also used



→ We also need to incorporate information from next words.

– We will address it letter in bi-directional RNN

* Forward Propogation

$\Rightarrow a^{<0>} = \vec{0}$

$\Rightarrow a^{<1>} = g(W_{aa} a^{<0>} + W_{ax} x^{<1>} + b_a)$ ← tanh more popular in RNN

$\hat{y}^{<1>} = g(W_{ya} a^{<1>} + b_y)$

→ sigmoid (binary)

softmax (k-way)

$\Rightarrow a^{<t>} = g(W_{aa} a^{<t-1>} + W_{ax} x^{<t>} + b_a)$

$\hat{y}^{<t>} = g(W_{ya} a^{<t>} + b_y)$

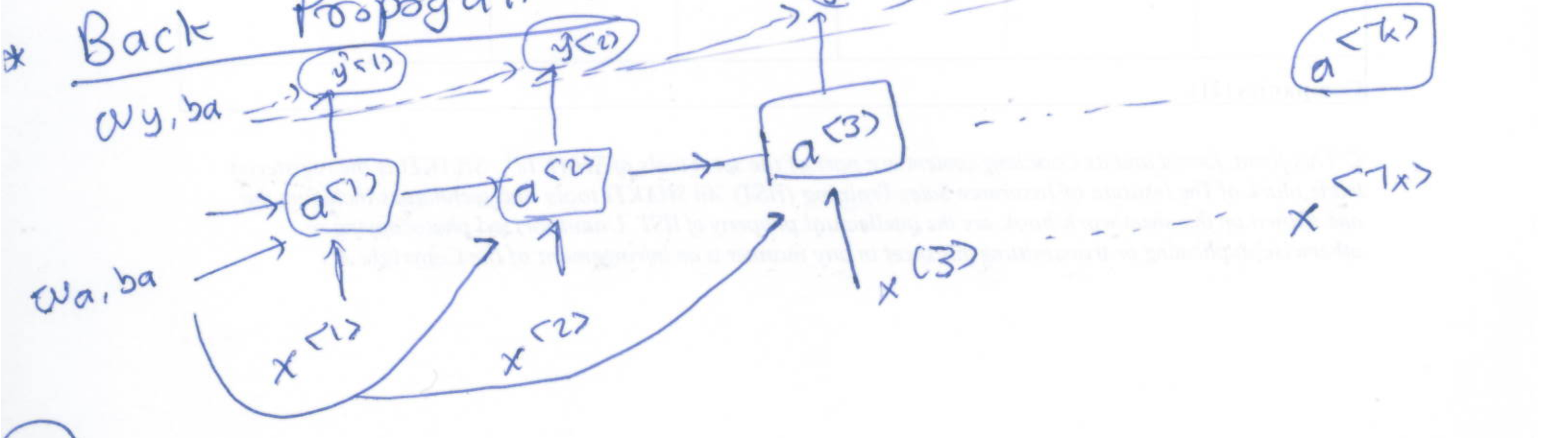
$\Rightarrow a^{<t>} = g(W_a [a^{<t-1>}, x^{<t>}] + b_a)$

where $W_a = [W_{aa} \,\vdots\, W_{ax}]$

$[a^{<t-1>}, x^{<t>}] = \begin{bmatrix} a^{<t-1>} \\ \hline x^{<t>} \end{bmatrix}$

$\hat{y}^{<t>} = g(W_y a^{<t>} + b_y)$

* Back Propogation



⇒ Cross entropy loss

$$L^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>} \log \hat{y}^{<t>} - (1 - y^{<t>}) \log (1 - \hat{y}^{<t>})$$

$$L(\hat{y}, y) = \sum_{t=1}^{T_y} L^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

⇒ Backpropogation through time

* Different types of RNN

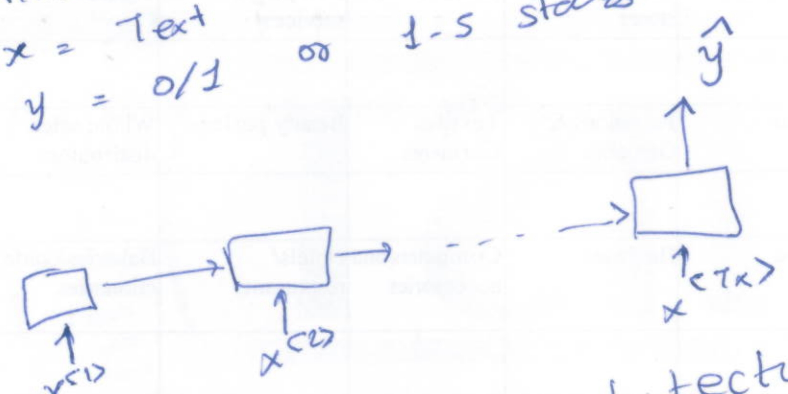
① $T_x = T_y$

- we were talking about this case so far architecture.
- Many to many

② sentiment classification

$x$ = Text

$y$ = 0/1 or 1-5 stars



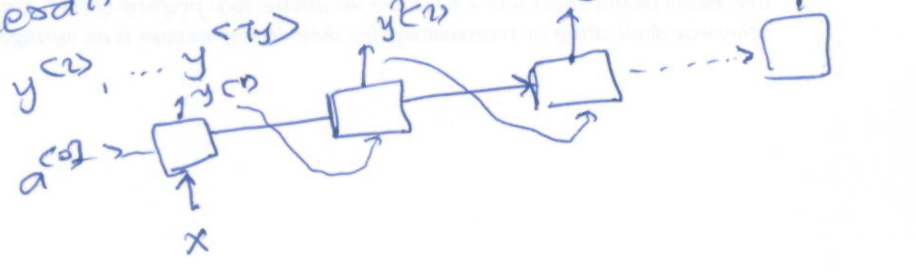

- many to one architecture

③ one to one

- Not much interesting
- standard general NN

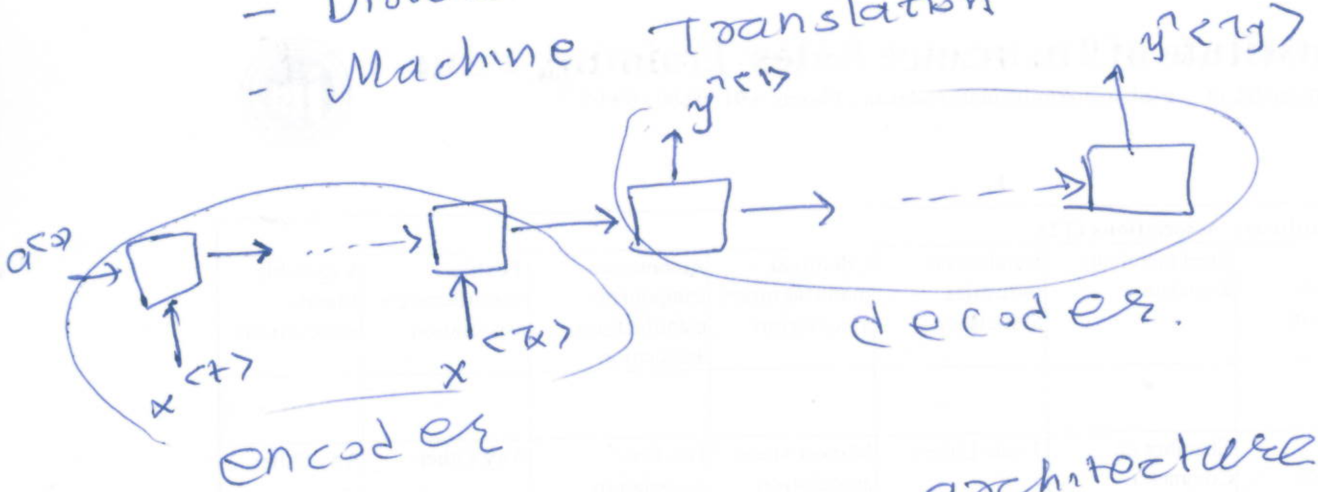
④ one to many

- Music generation
- $x \rightarrow y^{<1>}, y^{<2>}, \ldots y^{<T_y>}$



(5) $T_x \,!= T_y$

- Disserent length
- Machine Translation




- Many to many architecture

## * Langunge Modeling

=> P(sentece) = ?

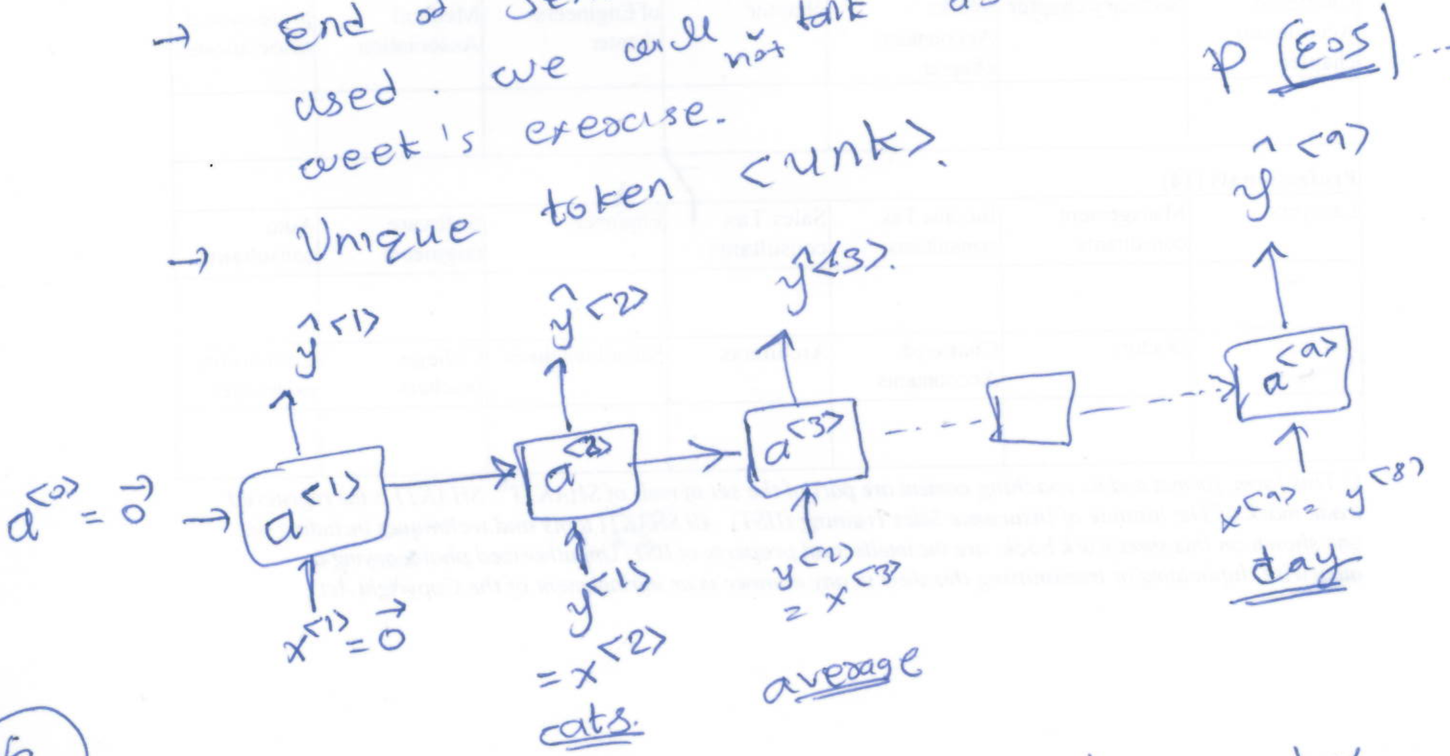
-> Fundamental component of speech recognition and machine translation

-> $P(y^{<1>}, y^{<2>}, \ldots y^{<T_y>})$.

-> Traing set: large coopus of English set

-> End of Sentence <Eos> taken can be used. we will not talk about it in this week's exercise.

-> Unique token <unk>.




(4)

- cats avergage 15 houes of sleep a day

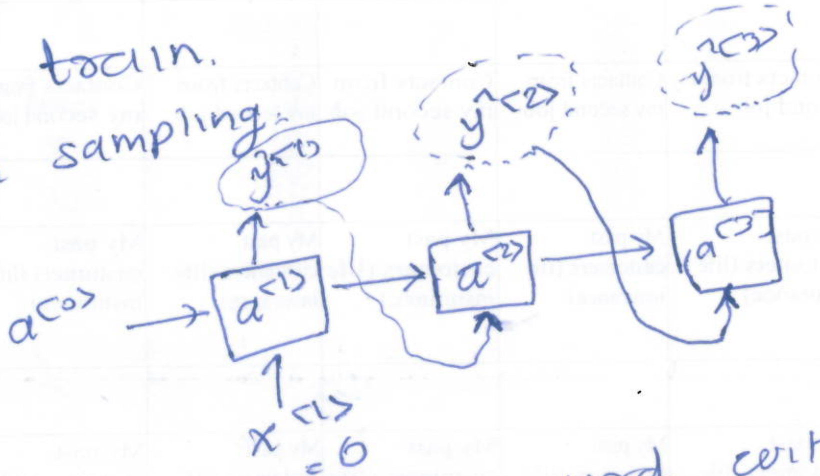
$$\Rightarrow \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$$

$$L = \sum_t L^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

$$P(y^{<1>}, y^{<2>}, y^{<3>})$$
$$= P(y^{<1>})\, P(y^{<2>} \mid y^{<1>})\, P(y^{<3>} \mid y^{<1>}, y^{<2>})$$

## * Sample Novel sequences

- First train.
- Then sampling.




- $\hat{y}^{<1>}$ → chose the word with highest probability
- If <unk> comes ignore it.
- When <EOS> comes sentence has been generated.

- We can also build character level language model.

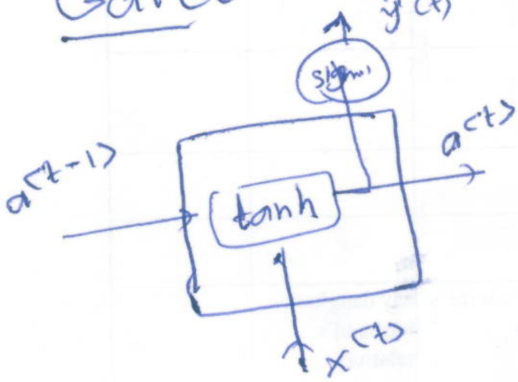
  Pros: - don't need to worry about unknow token.

  Cons: - Much longer sequences.
  - Not used generally.

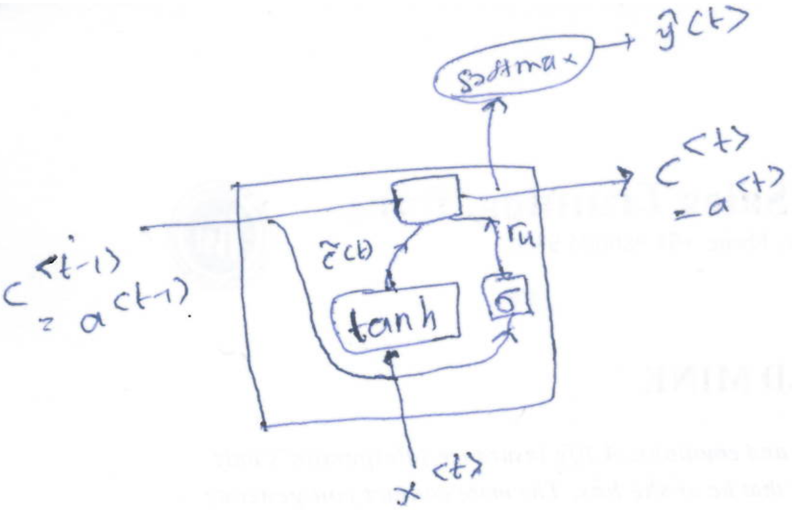
* <u>Vanishing gradient</u>

- The $\boxed{\text{cat}}$, ....., ....., ....., $\boxed{\text{was}}$ full
- The cats, ....., ....., ....., were full
- For DNN, back propagation can have little impact on earlier layer
- Idea is to have longer length dependency.

⇒ <u>Gated Recurrent Unit (GRU)</u>




- notion of cell c = memory cell
  - To remember whether it was cat or cats
  - $c^{<t>} = a^{<t>}$
  - It will be different LSTM
  - $\tilde{c}^{<t>} = \tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c)$
  - Gate $\Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>} + b_u)$
    - Intuition: think it as 0 or 1
  - Gate tells when to update $c^{<t>}$
  - $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$




⇒ $\Gamma_u$ is generally close to zero and hence mostly $c^{<t>} = c^{<t-1>}$ it helps with vanishing gradient

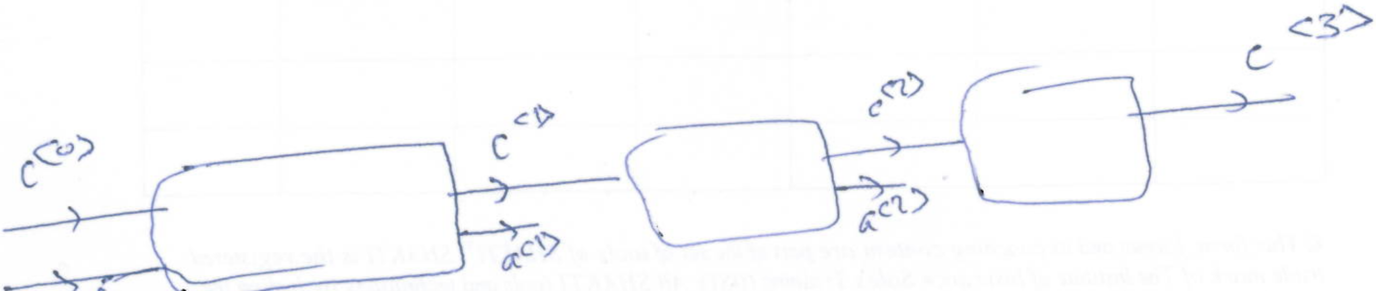
- In this way

⇒ if you have too activation dimensional $c^{<t>}$, $\Gamma_u$ will also be too dimensional

* <u>LSTM</u>

- we don't have $a^{<t>} = c^{<t>}$ here
- Two gates $\Gamma_u$ and $\Gamma_f$ ($\Gamma_f$ serves as $(1-\Gamma_u)$)
- one more output-gate $\Gamma_o$
- $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$
- $a^{<t>} = \Gamma_o * c^{<t>}$

u - update
f - forget
o - output

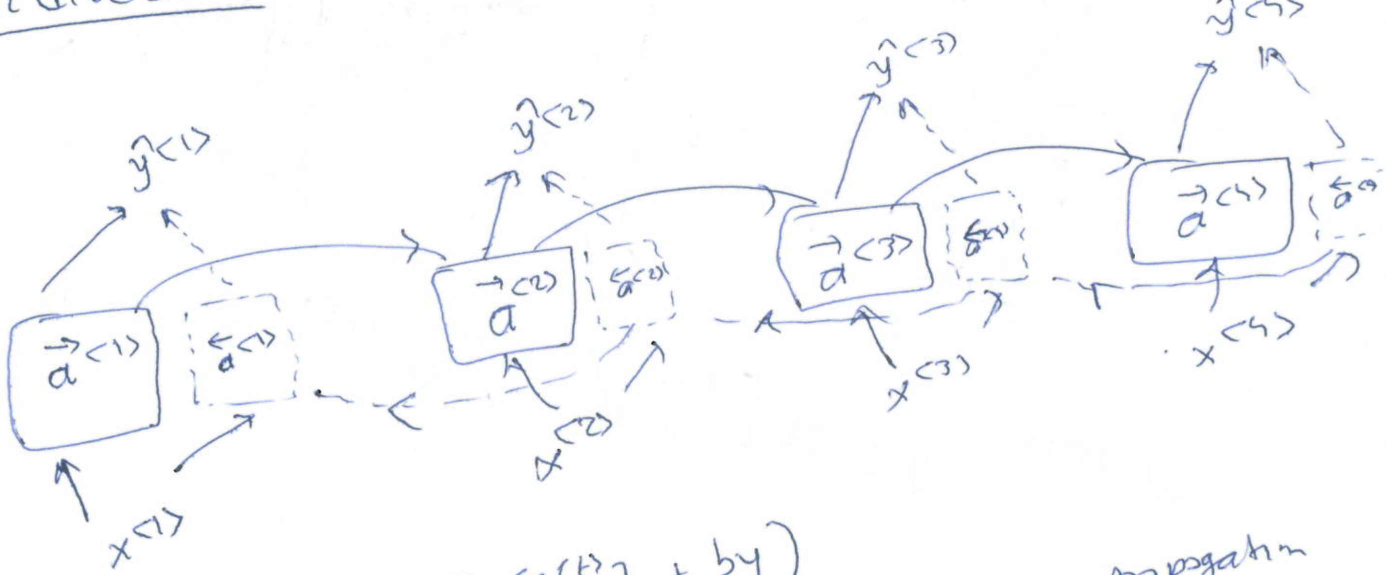



- There are some variation

7

- GRM is more faster
- LSTM is more powerful
  - people use it as default first thing to try.

* Bidirectional RNN (BRNN)




$$\hat{y}^{<t>} = g(W_y[\vec{a}^{<t>}, \overleftarrow{a}^{<t>}] + b_y)$$

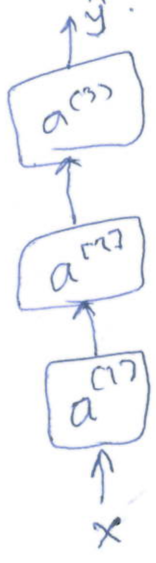
- This is still acyclic graph, forward propagation in backward direction

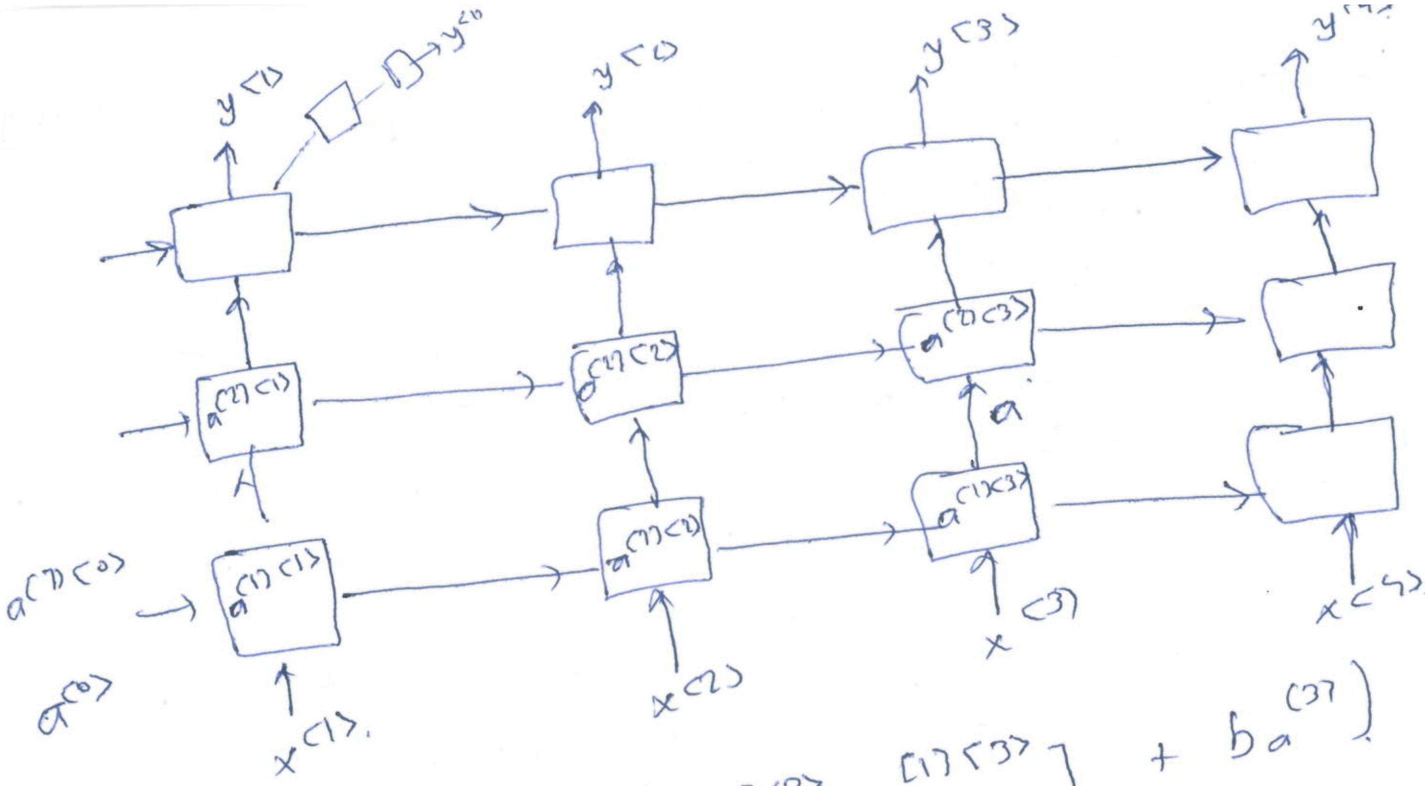
⇒ He said, Teddy Resault ........

- blocks can be RNN, GRU or LSTM

⇒ Disadvantage:
  - you need entire sequence of data

* Deep RNN






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

# NLP & Word Embedding

* Word Representation

- 1 hot vector
  - each token is independent
  - inner product of all words is 0
  - orange juice / apple juice
- Featurized representation
  - features for gender, royal, age, food
  - 300 features
- t-SNE algo
  - 300 dimension to 2 dimension and visulize

* Using word embeddings

⇒ Named entity recognition
  - learn embedding from billion words
  - transfer this knowledge to task illeed named entity where traing data is sample
    - Transfer learning
    - we also use less dimentional dense vector
- Generally used when you have relatively small dataset for task B.
- Relation to face encoding
  - getting vector for a face
  - Vector of activation of one layer of classifier

* Properties of word embeddings

- Man → woman as king → $\underline{\underline{?}}$
- Fing word $e_w$ : $\arg\max_w \; sim(e_w, e_{king} - e_{man} + e_{woman})$
- Above works in 300 D, not in 2D (after t-SNE)
  - t-SNE is non-linear

⇒ Similarity function
- cosine similarity → used often
- Euclidean

* **Embedding Matrix**
- getting vector from matrix (Multiply with one-hot)

$word = E \cdot O_{6257}$

$(300 \times 10{,}000) \cdot (10{,}000 \times 1)$

- while learning you initialize $E$ randomly and learn values via gradient descent

* **Learning word embeddings**
- Starting with historic models before going for word2vec or glove
- classifier to predict next word
- We will do repeated training
  - parameters to learn are $E$, $w^{(1)}, b^{(1)}, w^{(2)}, b^{(2)}$

⇒ context/target pair:
- context:
  - 4 words on left & right
  - last 4 words
  - last 1 word
  - Nearby 1 word → skip-gram (works well in practice)

orange ___?___

glass ___?___

useful in embedding

useful in language modeling

* <u>Word 2 Vec</u>

⇒ skip - grams :-

- context / target pair for supervised learning
- randomly pick up target in a window of ±5 words

$$O_c \rightarrow E \rightarrow e_c \xrightarrow{\quad} \underset{\text{softmax}}{\bigcirc} \longrightarrow \hat{y}$$

$$p(t/c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10000} e^{\theta_j^T e_c}}$$

$$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{10000} y_i \log \hat{y}_i$$

$y_i \rightarrow$ one hot vector

$\hat{y}_i \rightarrow$ probability of 10000 target words

⇒ <u>Hierarchical softmax</u>

- solves the problem of summing 10000 exponential
- In practice it does not use symmetric tree
  - more common words on top, less used words in deep.

⇒ <u>How to sample context c ?</u>

- the, of, a, and
- orange, apple, durian
- some heuristic to sample less frequent words equally.

⇒ Problem of skip-gram is to calculate denominator of softmax.

- solution is negative sampling

* Negative sampling

| context | word | target? |
|---|---|---|
| orange | juice | 1 |
| orange | king | 0 |
| orange | book | 0 |
| orange | the | 0 |
| orange | of | 0 |

$\underbrace{\text{context, word}}_{x}$ $\underbrace{\text{target?}}_{y}$

1 ← pick up from ± 10 window

0 ← pick up randomly

k = 5 to 20 smaller dataset

k = 2-5 for larger dataset

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

$$O_{6257} \rightarrow E \rightarrow e_{6257} \rightarrow \begin{cases} \circ \\ \circ \text{ juice} \\ \circ \text{ king} \\ \vdots \end{cases} \quad 10{,}000$$

- instead of 10,000 we will train 5 of them
- 10,000 binary classification is simpler than 10000 way softmax

→ Sampling negative sample
- Upon freq.
  - high chances of a, an, the.

- $$p(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{1000} f(w_j)^{3/4}}$$

* ## Glove

- Even more simpler,
  - slowly picking up but not as popular as word2vec
- $X_{ij}$ = # times $i$ appears in context of $j$
- $X_{ij} = X_{ji}$ or $X_{ij} \neq X_{ji}$ depending upon your context

$\Rightarrow f(X_{ij}) = 0$ if $X_{ij} = 0$
otherwise
= empercal weight to frequent / infrequent word

$$\text{minimize} \sum_{i=1}^{10000} \sum_{j=1}^{10000} f(X_{ij}) \left( \theta_i^T e_j + b_i + b_j' - \log X_{ij} \right)^2$$

$$e_w^{final} = \frac{e_w + \theta_w}{2}$$

because here e and θ plays symmetric role

* ## Sentiment classification

- Output is 5 star review
- You can quantify comments on social media as well
- Generally training data is less

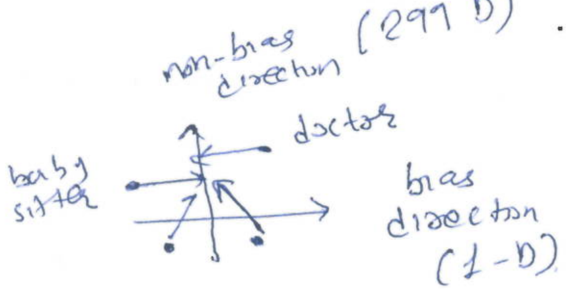
$\Rightarrow$ simple model
- get 300 D vector and average it
- and pass it to softmax classifier (1-5)
- Problem is it ignores word-order

⇒ RNN for sentiment classification

- Feed embedded vectors in RNN
- Many to one RNN

* Debiasing word embeddings

1. Identify bias direction




$e_{he} - e_{she}$

$e_{male} - e_{female}$

⋮

average

2. Neutralize
   - For words like doctor, baby-sitter, project them on non-bias direction

3. Equalize pair
   - grand-mother & grand father
   - move them to equidistance from non-bias direction

⇒ One researcher trained classifier to find out which words to neutralize or equalize

# Sequence models & Attention mechanism

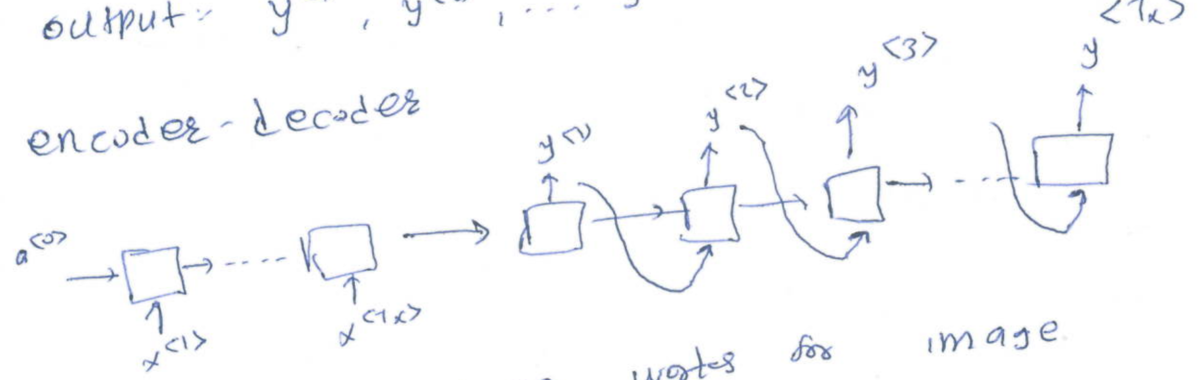
# Week-3

* **Sequence to sequence Model**

⇒ Input: $x^{<1>}, x^{<2>}, \dots x^{<T_x>}$

output: $y^{<1>}, y^{<2>}, \dots y^{<T_y>}$.

⇒ encoder-decoder




⇒ Similar architecture works for image captioning

- CNN followed by RNN

→ ~~You don't~~ want most ~~likely~~ sentence, but ra ~~best sentence~~

* **Picking most likely sentence**

→ In sentence generator we would pick up words based on their probability.

→ which is prediction $P(y^{<1>}, y^{<2>}, \dots y^{<T_y>})$

– Machine translation is "conditional language model", which models $P(y^{<1>}, y^{<2>}, \dots y^{<T_y>} \mid x^{<1>}, x^{<2>}, \dots x^{<T_x>})$

$$\arg\max_{y^{<1>}, \dots y^{<T_y>}} P(y^{<1>}, \dots y^{<T_y>} \mid x)$$

⇒ greedy search:
- Most likely first word

⇒ Exact search has. time complexity
- So we go for approximate search

## Beam Search

- Parameter beam-width, $B = 3$
- choose 3 most likely choices for first word.

→ keep top 3 choices after 2nd step as well
  - It is not exponential
  - After every step we have 3 choices only
- At every ~~step~~ step you have 3 copies of network
  - you need just 3 copies, not 30000 copies

→ setting $B = 1$, makes it greedy search

## Refinements to Beam search

⇒ Length normalization :-
  - Multiplying multiple probabilities make it a tiny no.
  - $\arg\max_y \prod_{t=1}^{T_y} P(y^{<t>} \mid x, y^{<1>}, \dots y^{<t-1>})$

$$\rightarrow \arg\max_y \sum_{t=1}^{T_y} \log P(y^{<t>} \mid x, y^{<1>}, \dots y^{<t-1>})$$

$$\rightarrow \frac{1}{T_y^{\alpha}} \sum_{t=1}^{T_y} \log P(y^{<t>} \mid x, y^{<1>}, \dots y^{<t-1>})$$

$\alpha = 1$ → full normalization
$\alpha = 0$ → No normalization
$\alpha = 0.7$ → can be tuned

⇒ How to choose beam width B ?

→ very large:-
  - better result
  - slower

→ small B :-
  - worse result
  - faster.
  - lower memory requirement

→ On production we see beam width around 10

## Error Analysis in Beam search

- where to spend time on ?
  - beam search or RNN

- Comput $P(y^*/x)$ and $P(\hat{y}/x)$ using RNN

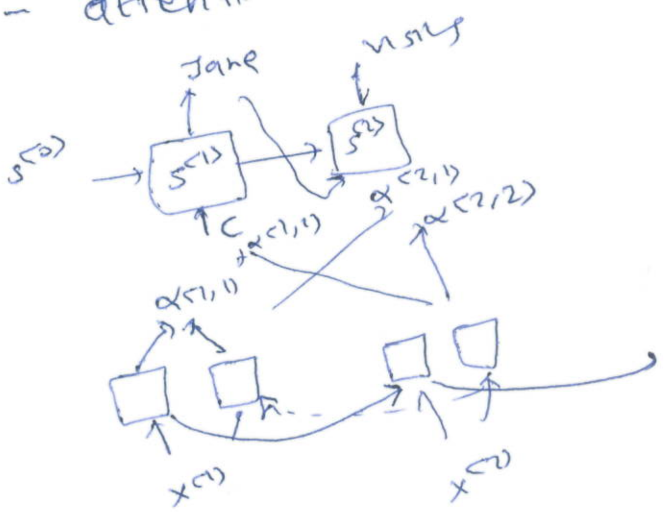
→ $P(y^*|x) > P(\hat{y}|x)$
  - Beam search choose $\hat{y}$
  - But $y^*$ attains higher $P(y/x)$
  - Beam search is at fault

→ $P(y^*|x) \le P(\hat{y}|x)$
  - $y^*$ is better than $\hat{y}$
  - But according to RNN it is worse
  - RNN is at fault

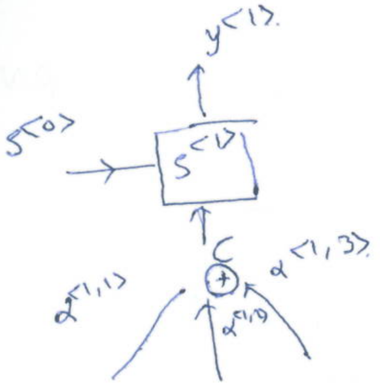
→ Carry error analysis for each sample in Dev set.
  - And see who is at fault more times

## * Attention Model.

- Human translator would work part by part
- encoder-decoder works well for smaller sentences
- attention model computes attention weight



$$a^{<t'>} = (\overrightarrow{a}^{<t'>}, \overleftarrow{a}^{<t'>})$$

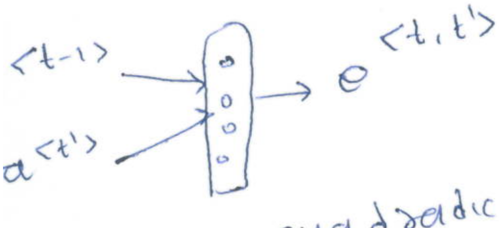



$$\sum_{t'} \alpha^{<1,t'>} = 1$$

$$e^{<1>} = \sum_{t'} \alpha^{<1,t'>} a^{<t'>}$$

$$a^{<t'>} = (\overrightarrow{a}^{<t'>}, \overleftarrow{a}^{<t'>})$$

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t'=1}^{T_x} \exp(e^{<t,t'>})}$$




Runs in quadradic cost.
$O(T_x \cdot T_y)$

→ Also applied in image captioning

→ Date in various format as input, standard format as output

→ visulization of attention weights

## Speech Recognition

MFCC in pre-processing

Attention model is one way

=> Another is CTC

- CTC = Connectionist temporal classification.

- # input timestamp $\overset{\text{much}}{>}$ # output timestamp

→ the quick brown fox

→ ttt _ h _ eee ___ ⌴ ___ qqq --

## Trigger Word Detection

- Alexa, Siri, Google Home
- 0/1 label for y
- It creates un-balanced training set
- few 1's when it triggered