ABSTRACTIVE TEXT SUMMARISATION

RNN
(Recurrent Neural Networks)

* used in speech recognition, language translation, stock prediction, spam mails, etc.

* Also used in image recognisation

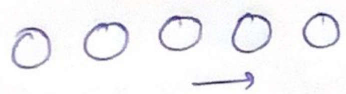
uses RNN
Apps
1. Google Img search
2. Img captioning

RNNs are neural networks
↓
that are Good at MODELLING SEQUENCE DATA.

Example

Ball — pic (still) — which direction would it move? — you can only Guess.

If



seems to be a SEQUENCE
i.e. ball is moving towards right

Sequence Data Forms (Input).

1. Audio → chunks, feed into RNN
2. Text → seq of char/word

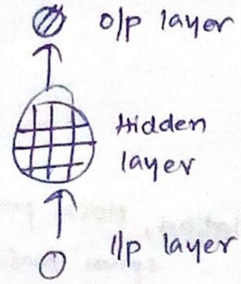
Sequential Memory

ABCD.. Z — easy ✓ — specific order/sequence

ZYXW.. BA — bit tough.

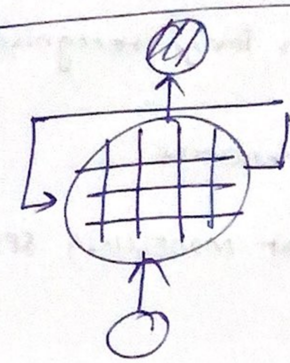
Mechanism that makes it easier for your brain to recognise sequence patterns.

Feed Forward Neural Network



inorder to use PREVIOUS INFO to AFFECT LATER ONES.

What if we have a loop?



RNN ✓.

Example

CHATBOT.

↓

classify intentions from user's IP text

↓

step1

Encode seq. of text using RNN.

step2

feed RNN's output to feed forward NN.

It'll classify intents → perform an action on screen.
used to

- start Activity
- send broadcast rec.
- start services
- send msgs b/w 2 Act.

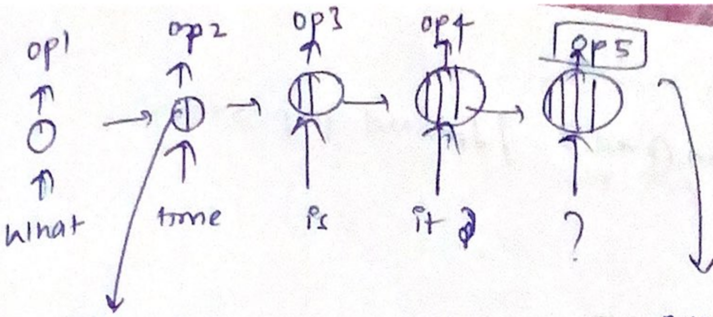
Ex: What time is it?

↓

What time is it ?

↓

Feed one word at a time to RNN sequentially.



Words will be converted to VECTORS of num/factors.

hidden state
↓
represents Info from all prev states

RNN encoded all data.

Asking for time.
↑
O feed forward NN
↑
op 5

code

```
rnn= RNN()
ff = Feed Forward NN()
hidden state = [0.0, 0.0, 0.0,0.0]  → shape dimensions

for word in input:
        output, hidden_state = rnn (word, hidden_state)

prediction = ff (output) .
```

problem

As it goes further, It has trouble retaining Info from prev. states

↓

SHORT TERM MEMORY & VANISHING GRADIENT } is due to the nature of back propagation

Train NN

step1 Make prediction (pred.) / forward propagaton

step2

compares pred with truth

$\boxed{\text{loss (pred, truth)}}$

↓

outputs ERROR VALUE

↓

estimates how badly network is performing

step3 back propagaton,

↓

calculates Gradients ∇ for each node in network

↓

value used to adjust networks internal weights allowing network to learn.

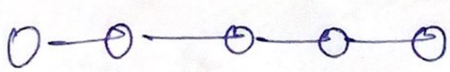
Bigger ∇, bigger adj.
vice versa.

prblm
∇ exponentially shrink.

Vanishing Gradient problem.

∇ is cal. wrt effects for each node. next node will have even smaller adj. if prev node has smaller adj.

RNN same prblm.



← ∇

early words are forgotten.

so, what to do ?

2 ways
- LSTM (long short term memory)
- GRU (Gated recurrent units)

learn to keep only relevant Info to make predictions

use GATES

diff tensor operations that can learn what info to add/remove to hidden states.

summary

* RNNs are good for processing sequence data for predictions but suffer from SHORT TERM MEMORY. Thus we also use LSTMs | GRUs.

* RNNs train faster, uses less computational resources.

why? because there are less tenser operations to compute.

The AI hacker - Michael phi

Krish Naik

Tut-29

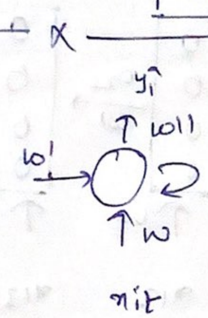
Why RNN

works WELL with sequence of data as inp

→ Time forecasting

→ stock forecasting, etc.

→ spam email or not



I/p data = Text data

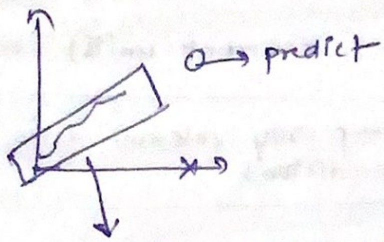
converted to VECTORS

word1	w2	. .	wn
1	0.75		2

seq. Info is imp.

And finds if sentence > 0 or sentence < 0

Time series (uses RNN)



Considers this entire time block of data

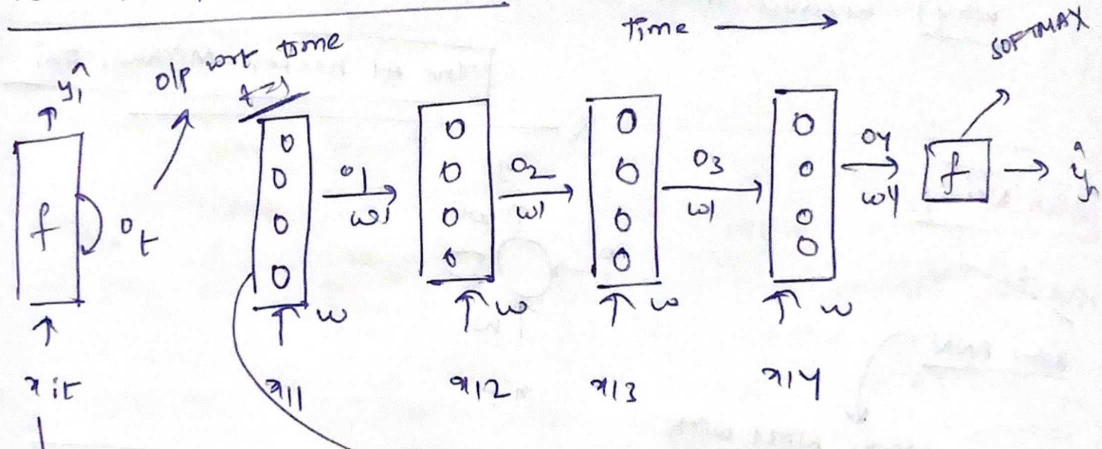
Applications

1. Google Image search
   - text converted to Image
2. Google lens
   - Image to text
3. Googe translator
   - many-many RNN.

seq. Info is kept.

Tut-30

Forward Propagation over time



Any no of dimensional features

HIDDEN LAYERS

⊛ Activation func. are used

Sentence $x_1 = \langle x_{11}, x_{12}, x_{13}, x_{14} \rangle$

At t=1 preprocess this to RNN

y_1, o_t, both sent to next RNN

∴ seq. Info is kept.

some func. (Act).

t=1 $O_1 = f(x_{11} \times w)$

t=2 $O_2 = f(x_{12} \times w + O_1 w')$

t=3 $O_3 = f(x_{13} \times w + O_2 w')$

t=4 $O_4 = f(x_{14} \times w + O_3 w_1)$.

$\hat{y}_a \rightarrow$ predicted value.

$\boxed{\text{loss} = (\hat{y} - y)}$

AIM: To reduce This

Backward propagation

LSTM (Michael - Phi) → The AI hacker.

- tanh → ensures val b/w -1, 1.
- sigmoid → 0 to 1.

$\boxed{\text{So, problem with RNN = Vanishing Gradient}}$

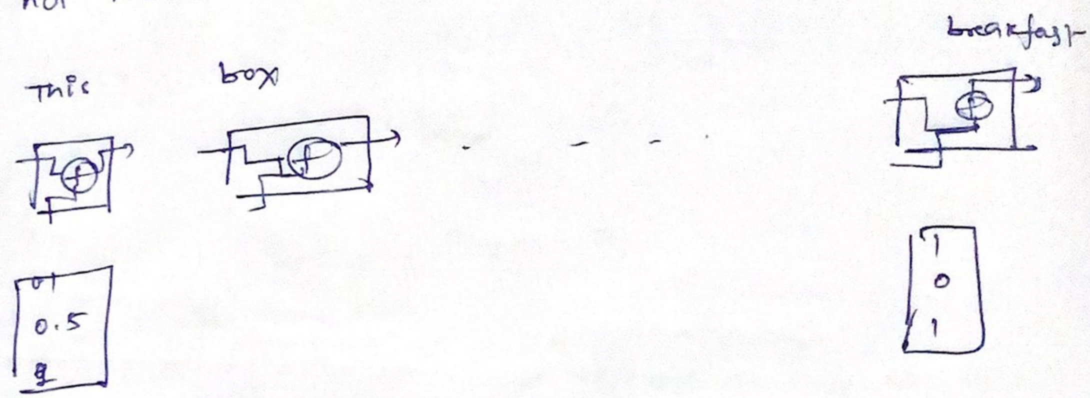
Rule: $\boxed{\text{new Weight = Weight - learning rate * Gradient.}}$

Example

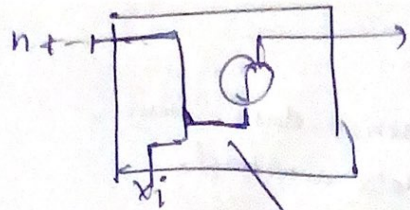
Amazing! The box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!

↓

Brain subconciously remembers only Imp. keywords like Amazing. not the, gave, etc.



- Words get transformed to machine readable vector.
- Then, each vector is processed to a RNN in a seq.
- While processing it passes PREV HEAD STATE to next seq in HIDDEN STATE
  → holds Info on prev data.



[ht-1 + xi] vector → has Info about curr, prev input

↓

goes to ACTIVATION fun.

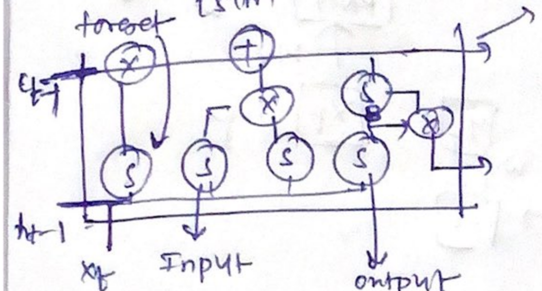
↓

output: ht hidden state

Here, tanh is used. (-1,1) → always.

Y SIGMOID is prefarable. (0 to 1). $\boxed{\text{used}}$ ✓.

LSTM.



→ cell state. (Memory) of network.

$\boxed{\text{LSTM}}$ $\boxed{\text{LSTM}}$

S → SIGMOID.

0 → forget
1 → keep

① ft * ct-1 = B.

like

find

$\boxed{\text{ct ✓. ht ✓.}}$

for ip in inputs:
ct,ht = LSTMcell (ct,ht, ip)

code

```
def LSTMcell (prev_ct, prev_ht, input):
    combine = prev_ht + input
    ft = forget_layer (combine)
    candidate = candidate_layer (combine) → C̄
    it = input_layer (combine)
  ✓ ct = prev_ct * ft + candidate * it
    ot = output_layer (combine)
    ht = ot * tanh (ct)
    return ht, ct.
```

The semicolon

Problem with Feed Forward NN

→ not designed for sequence / time series data. Hence, results with time series / sequential data are bad.

→ Does not model memory

Type of NN designed for capturing Info from sequence / time series data.

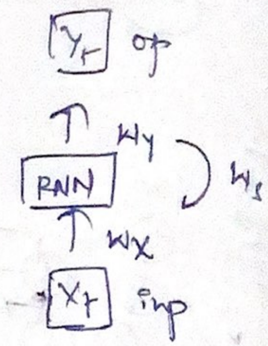
↑

simple RNN

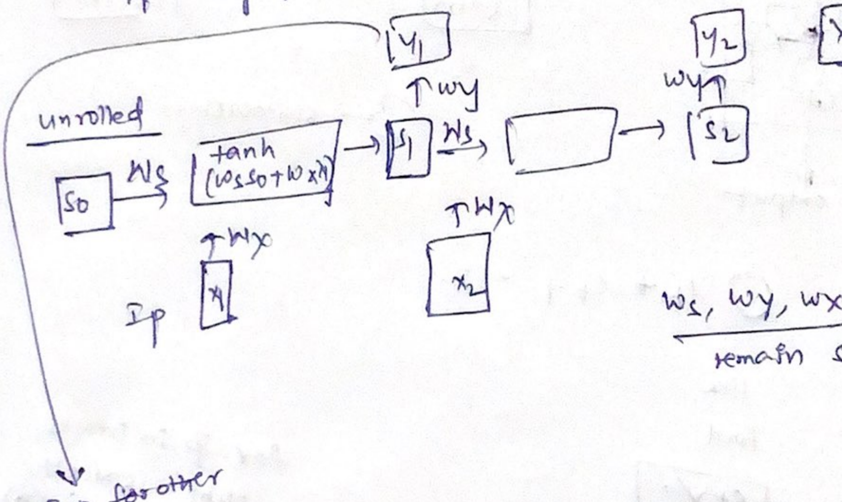
$$\boxed{s_t = F_w(s_{t-1}, x_t)}$$

$$s_t = \tanh(w_s s_{t-1} + w_x x_t)$$

$$y_t = w_y s_t$$



unrolled



w_s, w_y, w_x remain same.

→ Multiple layer RNN.

* loss cal. in backward.
* RNN learn in backward.

say, each state has $gr = 10^{-2}$.
100 states

1st state, $= (10^{-2})^{100}$
$\simeq 0$.

$\boxed{\text{update in states} = 0}$ ∴ neural network won't learn at all

} vanishing Gradient problem.

LSTM

$\boxed{\text{3 Gates + 1 cell state}}$

$f_t = \sigma(W_f s_{t-1} + W_f x_t)$ ⟶ forget Gate

$i_t = \sigma(W_i s_{t-1} + W_i x_t)$ ⟶ Input Gate

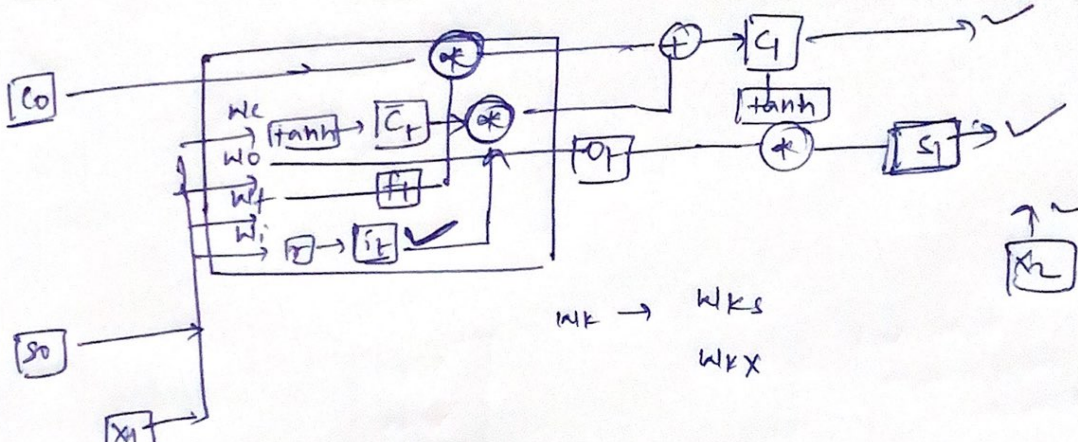
$o_t = \sigma(W_o s_{t-1} + W_o x_t)$ ⟶ output Gate

(*) Each gate has DIFFERENT WEIGHTS.

$$\boxed{\hat{c}_t = \tanh(W_c s_{t-1} + W_c x_t)}$$

$c_t = (I_t * \hat{c}_t) + (f_t * c_{t-1})$ ⟶ cell state.

$h_t = o_t * \tanh(c_t)$ ⟶ new state.



$W_k \rightarrow W_{ks}$

W_{kx}

operations

1. calculate i_t
2. calculate $\bar{c}_t$
3. $i_t * \bar{c}_t = \alpha$ $\boxed{\alpha + \beta = c_t}$
4. calculate f_t
5. $f_t * c_{t-1} = \beta$
6. calculate o_t
7. $o_t * \tanh(c_t) = \boxed{\text{new state} = s_t}$