Carse 5:-

RNN

Regular NN: A regular network take ~~out~~ input and gives output and then forget it.

RNNs are different they remember things over time

hidden state

Key trick: An RNN has a ~~layers~~ that gets passed from one step to the next. carring information from the past.

$$\text{tanh/Relu} \rightarrow a^{[1]} = g(W_{aa} a^{[0]} + W_{ax} x^{[1]} + b_a]$$

$$\hat{y}^{[1]} = g_a(W_{ya} a^{[1]} + b_y) \text{ sigmoid}$$

Normal equation:

$$a^{[t]} = g(W_y \, a^{(t-1)} + W_{ax} x^{[t]} + b_a)$$

$$y = g(W \, a^{[t]} + b_y)$$

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

Backpropagation through time in RNN:

Forward propagation

$$a^{[0]} \longrightarrow a^{[1]} \longrightarrow a^{[2]} \longrightarrow a^{3} \;\dots\dots\; a^{[T]}$$

$$x^{[1]} \qquad x^{[2]} \qquad x^{[3]} \qquad x$$

Loss

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

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

T#

Types of RNNs

One to one (One input, one output)

e.g Image → class

☆ One to many (One input, many output) Music Generation

~~One in~~ Image ⟹ [Word 1], [Word 2], [word 3]

☆ Many to One (Many Inputs, one output)

Word + Word + Word ⟹ [label]

* Many to many (Same length)

Word1 → Word2 → Word3 ⟹ Tag1, Tag2, Tag3

* Many to many (different length)

Input: Word 1 → Word 2 → Word 3

$\underbrace{\hspace{6cm}}_{\text{encoded Context}}$

↓

output

[Mot 1] → [Mot 2] → [Mot 3] → [Mot 4]

e.g

English to urdu translation

LANGUAG MODEL and RNNS.

Suppose: I said The apple and pear salad.

Language model: what did he say?

$P(\text{The apple and pear salad}) = 3.2 \times 10^{-13}$

$P(\text{The apple and pair salad}) = 5.7 \times 10^{-10}$

Solution:

Language model select sentence with greater probability

Corpus = Tokenize

How it works
Training set: large corpus of english text
tokenize
Cats average 15 hours of sleep a day.
(tokenize it)

RNN as a model language:-
A language model predicts the probability of the next token (word, subword or character) given the previous one.

Suppose: We have a large language model
How it will correct the sentence.
Cats average 15 hours of sleep a day
RNN tokenize this senctence:
⇒ and measure probability of every token (word, subwords)
⇒ It than arrange or predict words with best probabity.

like:

$$I \begin{cases} \rightarrow love = proba(0.9) \checkmark \\ \rightarrow kill = proba\ (0.5) \\ \rightarrow nonsense = proba(0.1) \end{cases}$$

then

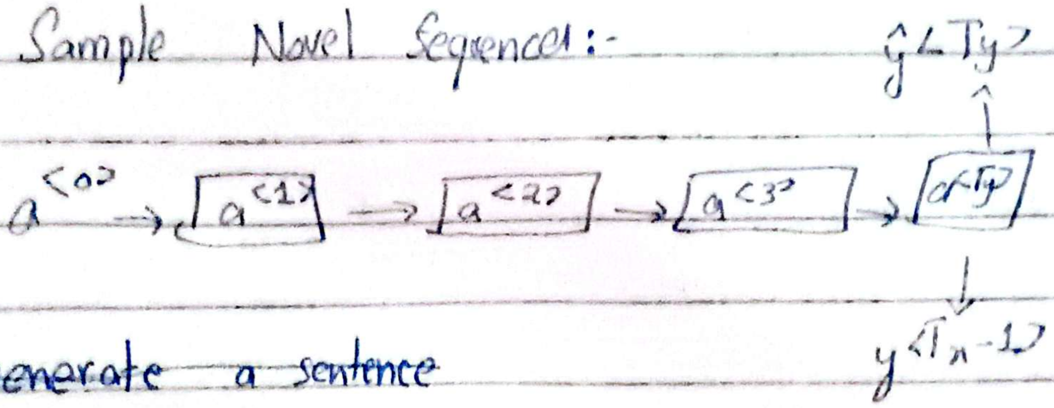
$\boxed{\text{I Love}}$ Answer.

Loss function:

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

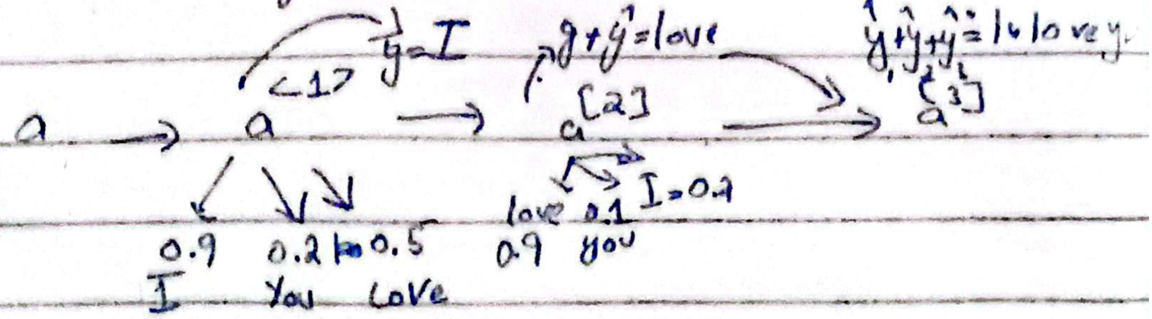
$$\mathcal{L} = \sum_t \mathcal{L}^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

To Sample Novel Sequences:-



Generate a sentence

I you love



The RNN select the word with higher probability. (Answer: I love U)

Vanishing Gradients ~~Problem~~ Problem:-

⇒ Varnishing gradient Probelem is hard to deal.

⇒ Exploding gradients can fixed by clipping

Vanishing gradient problem:-

Happens during training when gradients (the signal used to update weights) become smaller and smaller as the are propagated backward through many layers.

Consequence

- The RNN remembers short term patterns
- It forgets long-term context because the weights for earlier time steps hardly get updated.

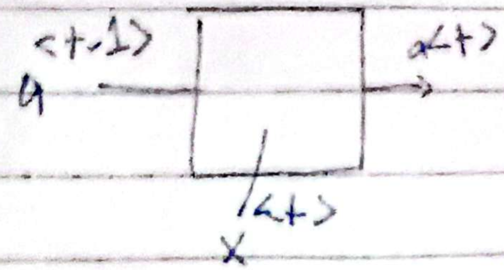
Fixing:-

- GRU
- Relu instead of tanh / sigmoid
- Residual connections in deep RNNs.

Note:- used to treat vanishing gradient.

Gated Recurrent unit (GRU):

Capturing long memory cell and treating vanishing gradient



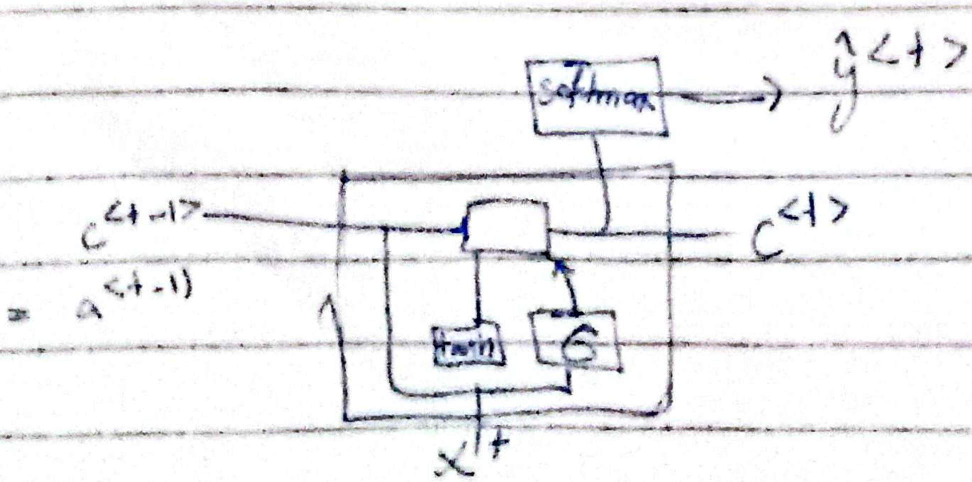
c = memory cell , $c^{<t>} = a^{<t>}$

$\tilde{c}^{<t>} = \tanh(W_c[c^{<t-1>}, x^{<t>}] + b_c$

gamma Γ_u = update gate = (0, 1)

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

Γ_u used to find singular or plural.



Equation

$$LSTM_s = c^{<t>} = \tanh(W_c [\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

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

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

Detailed

GRUs: simplify LSTMs by merging some gates, making them faster and lighter while keeping long-term memory.

GRU Gates:-

Update gate ⟹ decide how info of past to keep.
Reset Gate ⟹ decides how much of info to forget when computing new info.

Step 1 Update Gate: $z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$

$z \in (0,1)$

if close to 1 ⟹ keep more old memory
if close to 0 ⟹ Update more with new info

Step 2 : Reset Gate $\Rightarrow$ $r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$

if 0 $\Rightarrow$ forget old memory.
if 1 $\Rightarrow$ keep most of old memory

Step 3 : activation (new memory)

$\tilde{c}^{<t>} = \tanh \tilde{c}$

TH

GRU and Long-short-term-Memory (LSTM

Correction of equation

$a^{<t>} = \Gamma_o * \tanh(c^{<t>})$

GRU

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_c * c^{<t-1>}, x^{<t>}] + b_c)$$

Update gate

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

$$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r]$$

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

$$a^{<t>} = c^{<t>}$$

GRU $\tilde{c}^{<t>} = \tanh(W_c[\Gamma_c *$

LSTM

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

Update gate

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

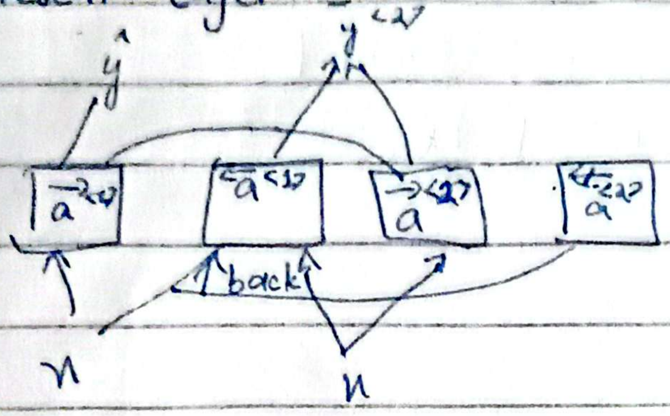
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$

LSTM $\tilde{c}^{<t>} = \tanh$

Bidirectional Infor

Take info from next and previous or present layer:



We can take info from future and past neuron cell.

☆ By taking info once in forward time and then in backward, this way it take info from future and past.

Working

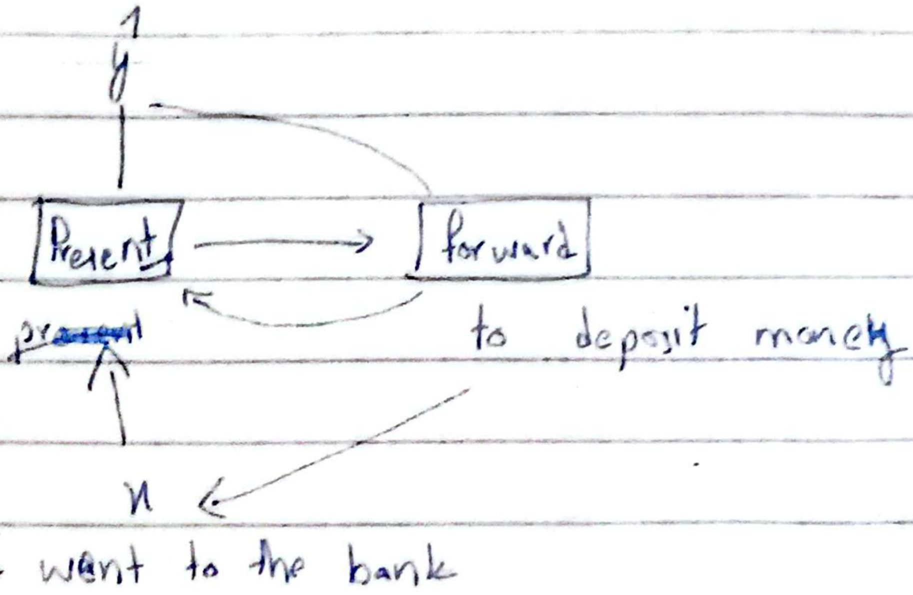
Trained Model

We want to check meaning

⟹ I went to the bank (financial or river bank?)

step 2: To check this we will go forward and backward in time.

Forward RNN: I went to the bank(iti a namy)
Back word RNN
see "to deposit money" (confirms it's a financial bank not a river bank).



III Deep RNNs
Stacking multiple RNN layers on top of each other like a sandwiched. to learn complex thing (NLPs; GAN)