Recurrent Neural N/w's

↳ feed forward neural N/w's rolled out over time

(Deal with Sequence data where the i/p have some defined ordering)

Several types of Architecture

① **vector - Sequence models**

(Take in a fixed size of vector as i/p) → Model → outputs a Sequence

Eg → Image Captioning

Vector Representation of an image (i/p) → Model → o/p Sequence of Sentence that describes the image

② **Sequence - Vector models**

Take in a Sequence as i/p → o/p fixed length vector

The main character sucked → [0.11 → good, 0.89 → bad] } The person tought about the movie

③ Sequence - Sequence Models

(Language Translation)

Takes in a Sequence as i/p → o/p's another Sequence

(Sentence in Spanish) → (Sentence in English)

RNN's → can also be used for Time Series Prediction

Disadvantages

① Slow to train

② Long Sequences → leading to vanishing/ exploding gradients

To overcome this problem they used LSTM N/w to introduce long memory to the N/w

(This allows us to retain memory for longer sequences)

① Normal RNN's → Slow

② LSTM's → Even Slower

(i/p data has to be passed sequentially/ serially one after the other → This doesn't make use of GPU's which can compute ||ely)

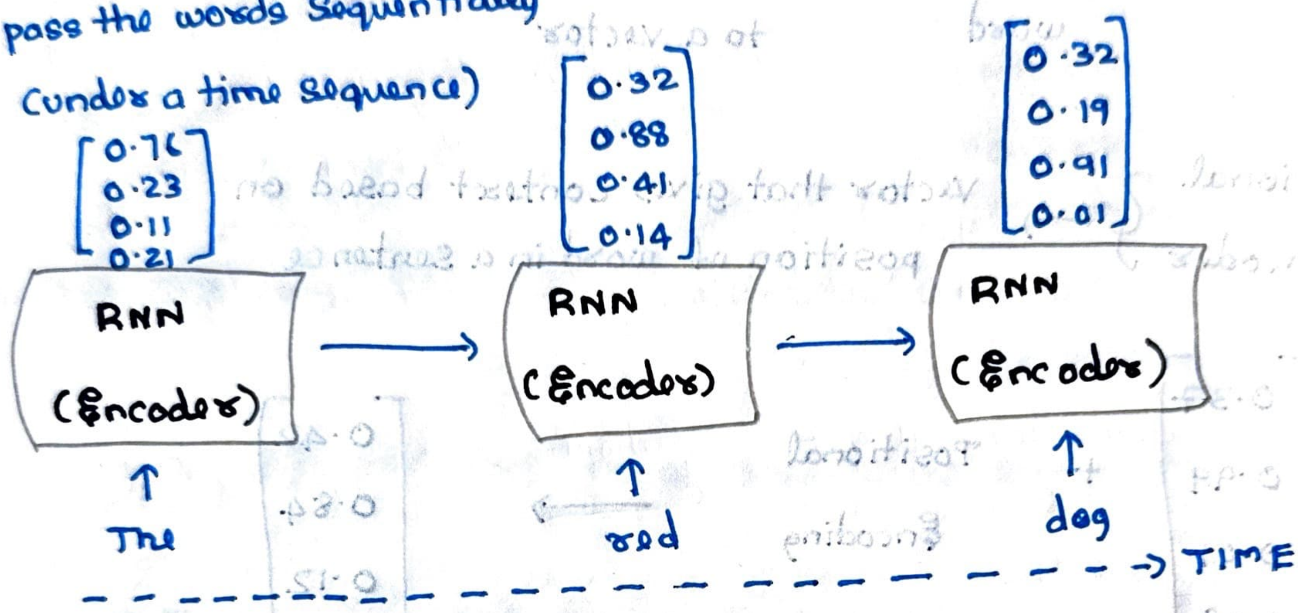
← we need the i/p of the previous state in order to perform any operation on the current state)

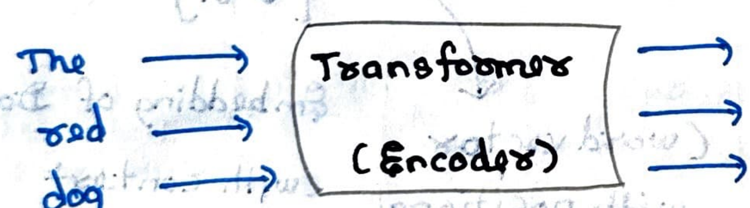
(Adv of Transformers)

① Sequences can be passed in ||el

Using RNN

we pass the words sequentially (under a time sequence)

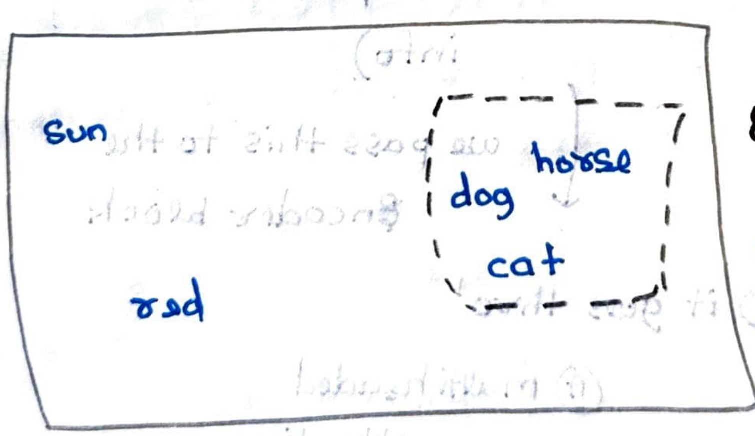




(||el vector o/p's)

(input Embeddings)

mapping every words to a space



Embedding Space (we can also pretrain this space to save time)

(where words with similar meaning are positioned close)

$\boxed{\text{Dog}} \longrightarrow \boxed{\text{Embedding Space}} \longrightarrow \begin{bmatrix} 0.37 \\ 0.99 \\ 0.01 \\ 0.08 \end{bmatrix}$

Maps a word to a vector

Positional Encoder $\}\longrightarrow$ Vector that gives context based on position of word in a sentence

$$\begin{bmatrix} 0.37 \\ 0.99 \\ 0.01 \\ 0.08 \end{bmatrix} + \text{Positional Encoding} \longrightarrow \begin{bmatrix} 0.42 \\ 0.84 \\ 0.12 \\ 0.81 \end{bmatrix}$$

(word vector with positional info)

Embedding of Dog (with context info)

we pass this to the Encoder block

1. it goes thro'
   A. multiheaded attention
   B. feed forward layer

(to tranform the Attention Vectors into a form that is digestible by the Encoder block)

Attention

↳ This tells us on what part of the i/p we must focus on?

(how relevant is the i^{th} word in a sentence w.r.t other words in the Sentence)

how relevant is the ith word in a sentence w.r.t other words in a Sentence → represented in the → ith Attention Vector (captures the textual relationship b/w words in a Sentence)

The → The big red dog $[0.71 \ \ 0.04 \ \ 0.07 \ \ 0.18]^T$

big → The big red dog $[0.01 \ \ 0.84 \ \ 0.02 \ \ 0.13]^T$

red → The big red dog $[0.09 \ \ 0.05 \ \ 0.62 \ \ 0.24]^T$

dog → The big red dog $[0.03 \ \ 0.03 \ \ 0.03 \ \ 0.91]^T$

→ 8 such attention vectors (and then we compute the weighted average) → hence we get the vector

Decoder

we get the french word

Le → convert this into vector (input embedding) → Positional Embedding → Pass it to the Decoder block

Each of the attention nets are independent of each other because of which we can pass them ||ely

Decoder → generates vector for every word in a Sentence

(Self attention is done for the french words)

Le → ~~Le~~ gros chien rouge

gros → Le ~~gros~~ chien rouge

chien → Le gros ~~chien~~ rouge

rouge → Le gros chien ~~rouge~~

(Attention vectors)

$$\underset{\text{Le}}{\begin{bmatrix}1\\0\\0\\0\end{bmatrix}} \quad \underset{\text{gros}}{\begin{bmatrix}0.1\\0.9\\0\\0\end{bmatrix}} \quad \underset{\text{chien}}{\begin{bmatrix}0.05\\0.40\\0.55\\0\end{bmatrix}} \quad \underset{\text{rouge}}{\begin{bmatrix}0.16\\0.09\\0.15\\0.66\end{bmatrix}}$$

(Another Attention block)

$$\boxed{\text{Encoder-Decoder Attention}}$$

$$\underset{\text{The}}{\begin{bmatrix}0.71\\0.04\\0.07\\0.18\end{bmatrix}} \quad \underset{\text{big}}{\begin{bmatrix}0.01\\0.84\\0.02\\0.13\end{bmatrix}} \quad \underset{\text{red}}{\begin{bmatrix}0.09\\0.05\\0.62\\0.24\end{bmatrix}} \quad \underset{\text{dog}}{\begin{bmatrix}0.03\\0.03\\0.03\\0.91\end{bmatrix}}$$

(Vectors from Encoder)

(how related each word vector is similar with respected to each other)

O/p of the block is the Attention vectors for every word in English & French Sentence

Each vector represents a relationship with other vectors in both the languages

(Then each attention vector is passed to a feed forward Neural N/w in order to make it more digestable by the next layer

① Linear ⟶ This has another feed forward layer (used to expand the dim to the no. of words in french language)

② Softmax ....)

↓

converts it into a prob dist

↓

final word is the one with the highest prob

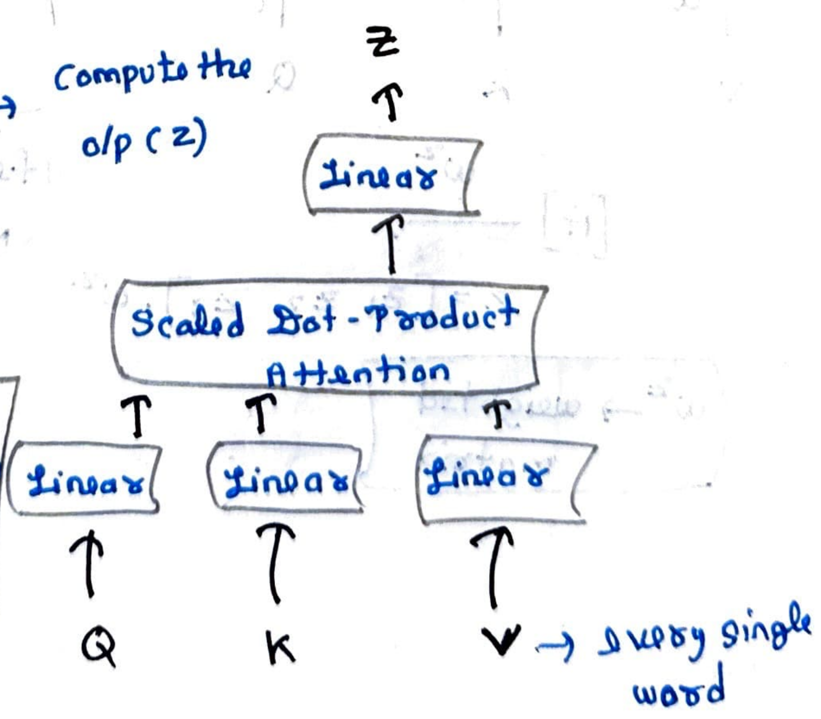
masked Attention block ⟶ (while generating the next french word we can use all the words from the english Sentence but only the previous word from the french Sentence)

we have Q, K, V } vectors for every single word ⟶ Compute the o/p (Z)

(Single headed Attention) ⟶

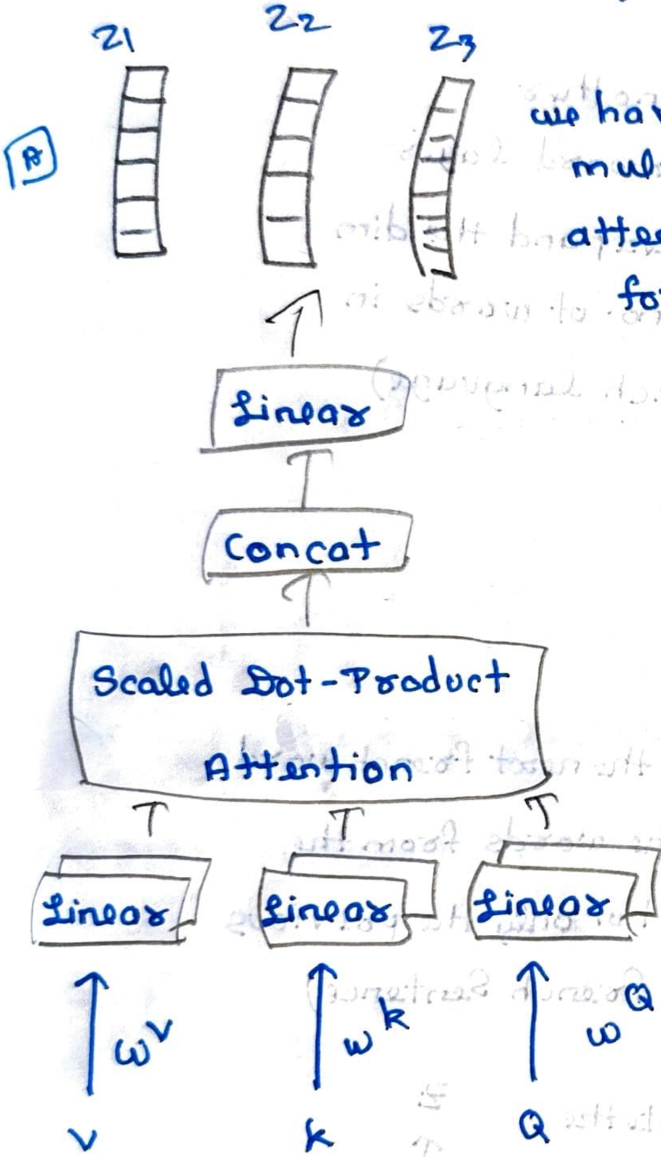
$$\boxed{Z = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{\text{Dimension of vector } Q, K \text{ or } V}}\right) \cdot V}$$

↓ Compute attention vectors for every word



for multi headed Attention

(we have multiple weight matrices)



we have multiple attention vectors (z) for every word

Same z formula

After each and every layer we perform some kind of normalisation (batch normalisation Layer normalisation)

↳ Read about this

$\boxed{A} \xrightarrow{w^z}$

$z = [z_1 \; z_2 \; z_3] \cdot w^z$

$w^z \rightarrow$ weighted matrix

feed forward Neural N/w

makes sure it is only one (z) per word attention vector