Gen AI with langchain & Huggingface

VS Code

Creating an environment

→ terminal → cmd →

conda create -n venv python=3.12 -y

Conda not recognized error

→ conda prompt → where conda

→ updak environment variables

→ check → cmd → conda --version

→ conda prompt → conda init cmd.exe

conda init powershell

→ pip install ipykernal

py

running a python file → python app.py

line continuation → 1+2+3+
4+5 \

x=5; y=6; z=x+y

break - exits whole loop
continue - skips that iteration
pass - does nothing

L = [ ] ; L[::] → all elements

L = [x**2 for i in range(5) if i%2 ==0]

L.extend([1,2,3,4]) → add 1,2,3,4 to the list

L.count(2) → count occurances of 2

L.index(2) → return 1 index of 2

L.remove(2) → removes first occurrence

L.insert (3, 99) → 3 is position

L*3 => [1, 2, 3, 1, 2, 3, 1, 2, 3]

L + 5 → error

t = 1, 2, 3 => (1, 2, 3)

dictionary = unique Keys & unordered items

dictionary

d = {'a': 1, 'b': 2, 'a': 5}

print(d) = a : 5
b : 2

accessing → d[a] => 1
d.get(a) = 1
d.get(d) = None

d.get('d', 'not avai')

d[d] = 'not available'

del d[a]

d_copy = d.copy() # shallow copy

d_copy → different memory

d.copy() → different memory

d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}

copy_d = d

d[c] = 9

copy_d[c] → 9 instead of 3

lambda fun^n → Single operation

→ add = lambda x, y: x+y

add(5, 2) ⇒ 7

m = map (lambda x: x^2 , [1, 2, 3, 4])

map applies a function to a list

m = 0x22fcd23 → reference

list(m) → [1, 4, 9, 16]

list (filter (lambda x: x%2==0 , [1, 2, 3, 4, 5]))

map, filter → returns object

for creating a package

package → folder name

↳ __init__.py with empty needs to be created

↳ maths.py → module

from package.maths import addition

folder ↑ module ↑ function ↑

```
with open("demo.txt", 'w+') as file;
    file.readliner()
```

---

os.listdir('.') → . means current dir

('../') → previous folder

os.path.join()

os.path.isdir()

os.path.isfile()

```
try:
    1/0
except e1:
except e2:
else: print(res)
finally: print(exec completed)
```

Classes → blueprint of objects

```
class Cars:

    def __init__(self, name, doors):

        self.name = name
        self.doors = doors      } instance variables, attributes

    def print_details(self): → instance method

        print(self.name, self.doors)

audi = Cars('Audi', 4)
audi.print_details()
```

Inheritance

```
class Audi(Cars):

    def __init__(self, name, doors, car_type)

        self.car_type = car_type

        super().__init__(name, doors)

        print(name, doors, car_type)

Audi('Audi', 4, 'Diesel')
```

```python
class Father:
    def __init__(self, money, experience):
        self.money = money
        self.experience = experience

    def print_father(self):
        print(self.money, self.experience)

class Mother:
    def __init__(self, care, values):
        self.care = care
        self.values = values

    def print_mother(self):
        print(self.care, self.values)

class Child(Father, Mother):
    def __init__(self, money, experience, care, values, name):
        Father.__init__(self, money, experience)
        Mother.__init__(self, care, values)
        self.name = name

    def print_child(self):
        print(f"""From father {self.name} received {self.money} amount and {self.experience},
              \nFrom Mother {self.name} received {self.care} and {self.values}""")

child_obj = Child(99999999, 10, 1000, 100000, 'Rahul')
child_obj.print_child()
```

✓ 0.0s

```
From father Rahul received 99999999 amount and 10,

From Mother Rahul received 1000 and 100000
```

Natural Language Processing

tokenisation

- Corpus — Paragraph
- Document — Sentences
- Vocabulary — Unique words

Hi I am Rahul. I am a very rich person

Paragraph / Corpus

tokenisation →

① Hi I am Rahul

② I am a very rich person

Sentences / Document

tokenisation

Words

Stemming

Lemmatize

Stop words

NER

POS

One hot encoding!!

	text	OP	Vocabulary size = 7 The food is good bad pizza ama
D1	The food is good	1	1 0 0 0 0 0 0 ← The
D2	The food is bad	0	0 1 0 0 0 0 0 ← food
D3	Pizza is amazing	1	0 0 1 0 0 0 0 ← is

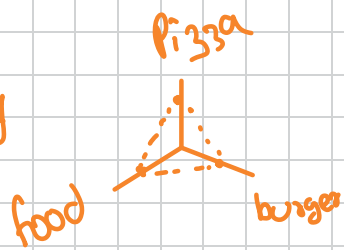
D1 [[1000000], [0100000], [0010000], [0001000]]

D2 [[1000000], [0100000], [0010000], [0000100]]

4x7

Disadvantages –

1. Sparse matrix → Leads to overfitting
2. No semantic meaning captured
3. No fixed size I/p
4. Out of vocabulary



Vectors for each word!!

Bag of words

	OP
He is a good boy	1
She is a good girl good	1
Boy & girl are good	1

lowercase & remove Stopwords

S1 → good boy

S2 → good girl good

S3 → boy girl good

Vocabulary	frequency
good	4
boy	2
girl	2

sort desc

	good	boy	girl
D1	1	1	0
D2	2	0	1
D3	1	1	1

Vectors for each sentence

OHE → Each word has its encoding

BOW → Each word has its frequency

TFIDF high frequency words gets lower importance

low frequency words gets higher importance

$$\text{TF (Term frequency)} = \frac{\text{frequency of word}}{\text{No of words in sentence}}$$

$$\text{IDF (Inverse document frequency)} = \log_e\left(\frac{\text{No of sentences}}{\text{No of sentences containing word}}\right)$$

S1 Good boy

S2 Good girl

S3 boy girl good

Term frequency

	S1	S2	S3
good	1/2	1/2	1/3
boy	1/2	0	1/3
girl	0	1/2	1/3

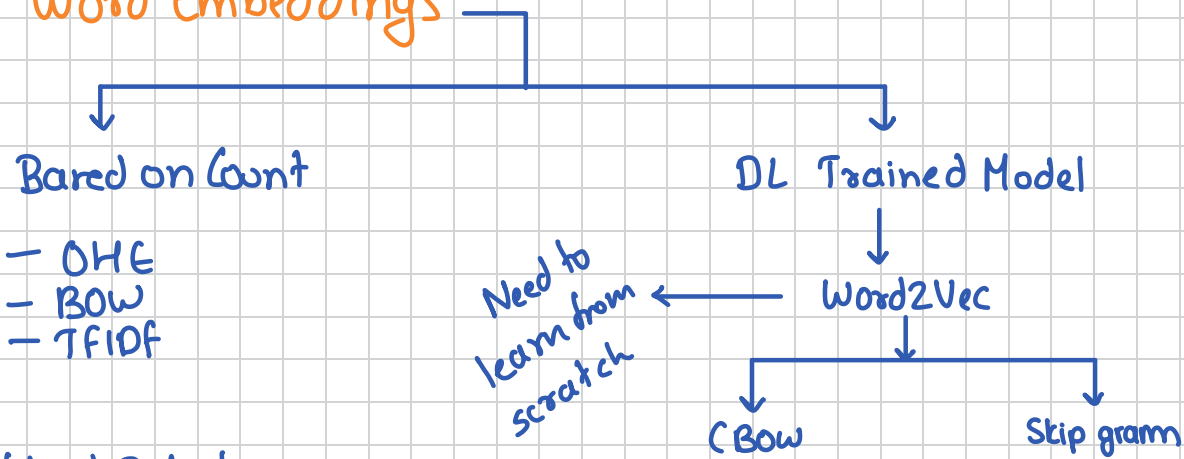
IDF

good $\log_e(3/3) \Rightarrow 0$

boy $\log_e(3/2) \Rightarrow$

girl $\log(3/2) \Rightarrow$

Word Embeddings



Word 2Vect

→ feature representation of every word

- NN to learn word association from a large corpus
- NN method to turn words into vectors
-

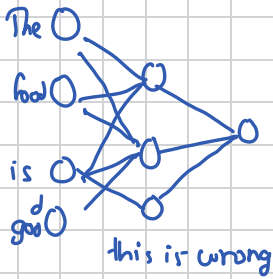
RNN VS ANN for text

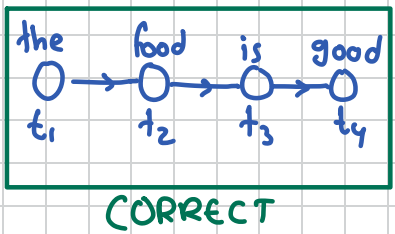
→ ANN for data is not correct.

→ ANN passes all information at once.

– Since text data is sequential data

– RNN is perfect for such scenarios





CORRECT

Simple RNN – It remembers

- A RNN is a NN with a memory or feedback loop.
- It deals with sequential data like audio, text, stock market.
- It processes each sentence word by word.

How it works ?

- At each step (word by word) –
  - Input at time $t = x_t$
  - Hidden state at time $t = h_t$ (memory)
  - Previous hidden state h_{t-1}

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$$

Important Points -

- RNN has hidden states that acts as a memmory
- Weights are shared across time (same params for each step)
- Good for sequential data

$h_t = W_x x_t + W_h h_{t-1} + b \quad x = [2, -1, 3]$

$W_x = 1$
$W_h = 0.5$
$b = 0$
$h_0 = 0$
$x_2 = [2, -1, 3]$

$h_1 = ?$ for $x_1 = 2$

$\Rightarrow t = 1 \quad x_t = x_1 = 2$

$h_1 = 1 \times 2 + 0.5 * 0 + 0$

$h_1 = 2$

$h_2 = W_x x_2 + W_h h_1$

$= 1 \times -1 + 0.5 \times 2$

$= -1 + 1$

$= 0$

$W_h = 0.5$
$W_x = 1$

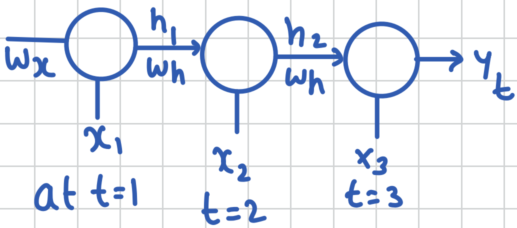
$h_t = W_x x_t + W_h h_{t-1}$

$h_3 = W_x x_3 + W_h h_2$

$= 1 \times 3 + 0.5 \times 0$

$= 3$

$$h_t = W_x x_t + W_h h_{t-1} + b$$



tanh for −1 to 1 to learn complex problems

$$W_{new} = W_{old} - \eta \frac{\partial L}{\partial W_{old}}$$

η - learning rate

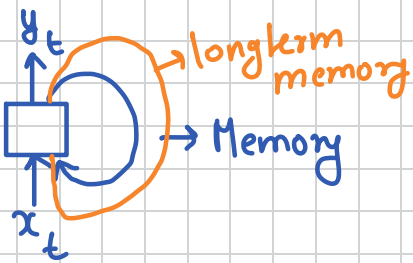
Problems with RNN

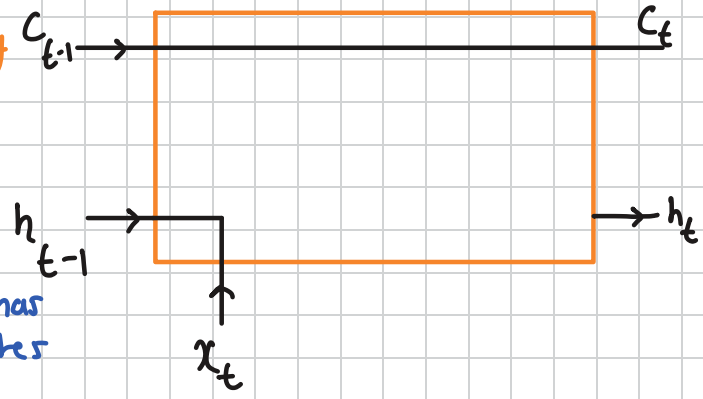
- long term dependency cannot be captured
- Vanishing gradient problem
- Exploding gradient problem

mid won
sprinklr close
labs so main switch
streamlit ko kaise

Long Short Term Memory -

Overcomes the problems of RNN
- long term dependency
- vanishing / exploding gradient





Basic idea is LSTM has 3 gates - 3 for cell states

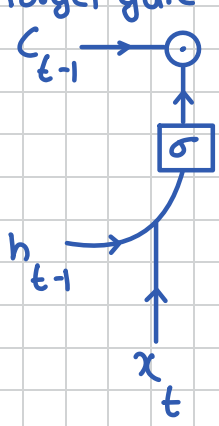
1. forget gate – What to forget
2. Input gate - What to add
3. Output gate – What to output

C_t is Cell state → long memory

h_t is hidden state

x_t is word input at current t

forget gate -

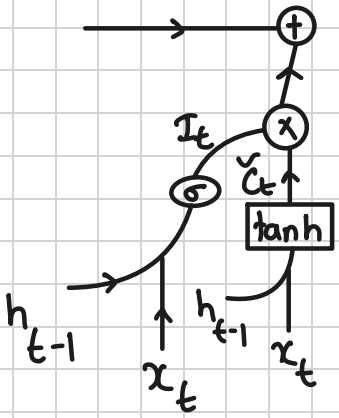


- This gate is responsible for forgetting context from cell state
- $\sigma[h_{t-1}, x_t] = [0\ 0\ 0]$ → forget everything
- $\sigma[h_{t-1}, x_t] = [1\ 1\ 1]$ → Keep everything

$$f_t = \sigma(w_f \cdot [h_{t-1}, x_t]) + b_f$$

2. Input gate

$C_{t-1} \times \sigma(h_{t-1}, x_t) \oplus \sigma([h_{t-1}, x_t]) \otimes \tan([h_{t-1}, x_t])$



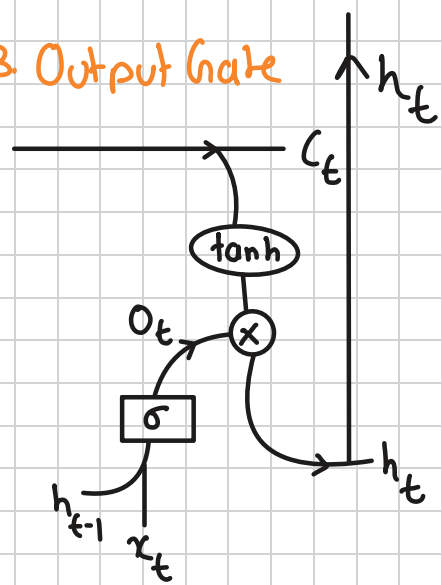
$$I_t = \sigma\left([W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

$$\tilde{C}_t = \tan\left([W_c \cdot [h_{t-1}, x_t] + b_c\right)$$

$\tilde{C}_t$ = Candidate memory → Add new memory

Bared on previous output of h_t ie h_{t-1} & current input, what input do we need to add to the cell state.

3. Output Gate



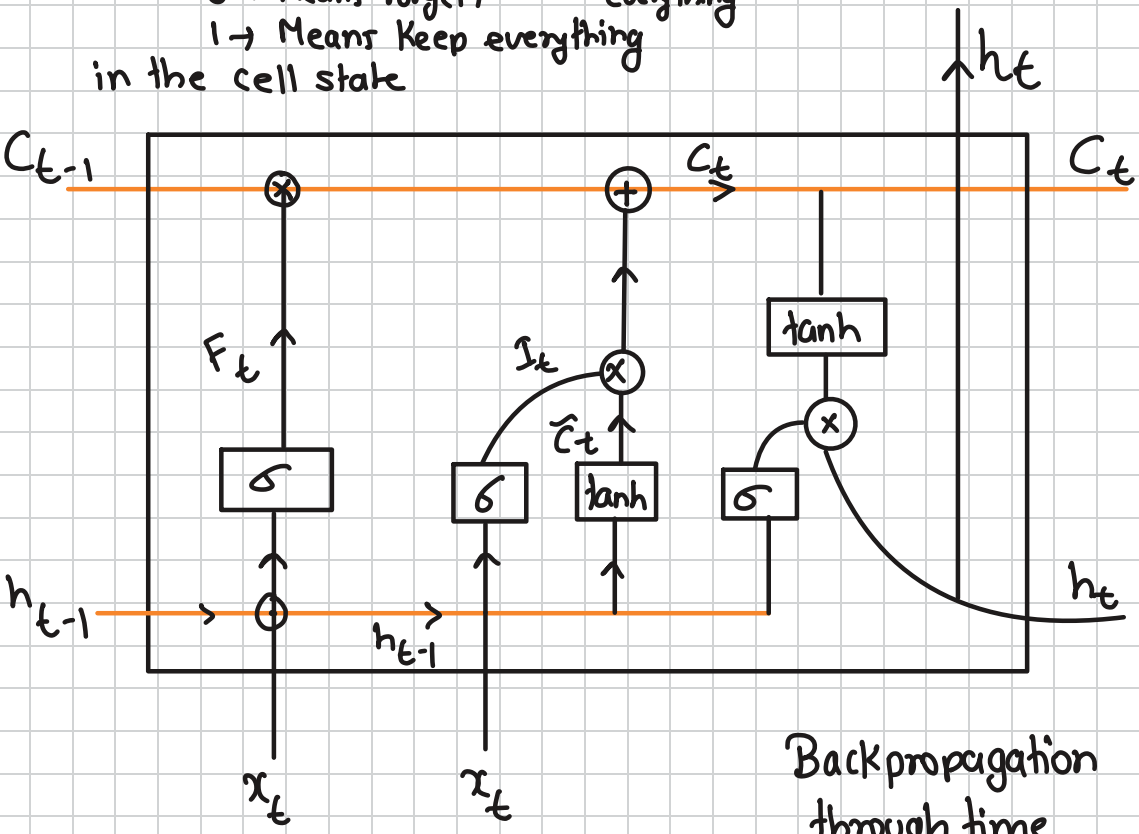
$$O_t = \sigma[W_o \cdot [h_{t-1}, x_t] + b_o]$$

$$h_t = \tan(W_c C_t + b_t) \times O_t$$

Value from $\sigma \rightarrow$ 0 to 1

0 → Means forget / everything

1 → Means Keep everything in the cell state



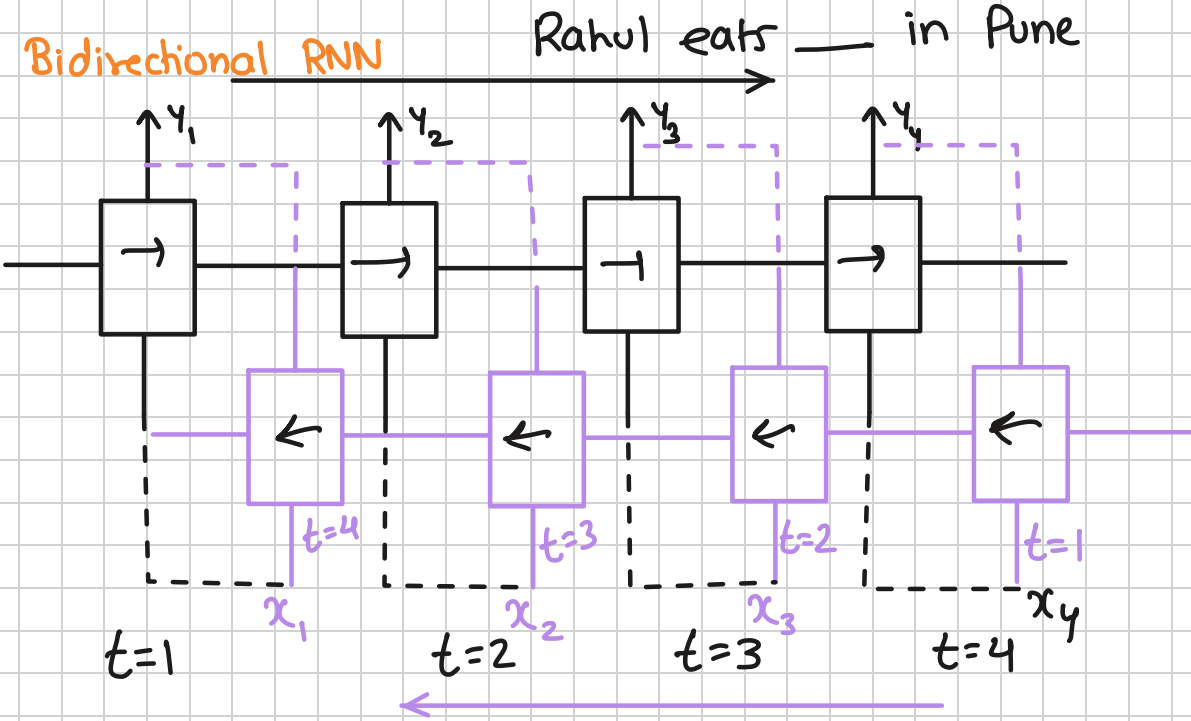
Backpropagation through time

Trainable parameters – Weights & biases

Gated Recurrent Network – left this topic for later

- Types of RNN –
  - Sequence to sequence
  - Sequence to vectors
  - Vectors to sequence
  - vectors to vectors

Bidirectional RNN



→ BiRNN's processes sequences in both the directions and combines the information

→ forward RNN → $x_1 \rightarrow x_T$, Backward RNN $x_T \rightarrow x_1$

→ Captures past and future context

→ Improves performance in sequence task

→ Cannot be used for streaming

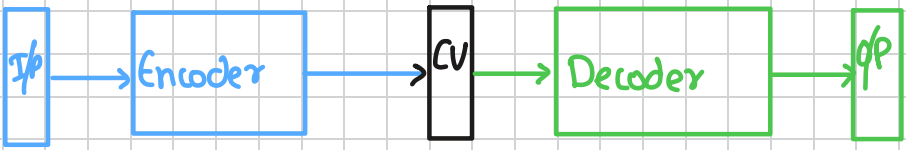
→ Heavy model, slower training

ex - POS Tagging, NER, Speech Recognition

Encoder- Decoder (Seq2seq)

→ for sequence to sequence use cases.

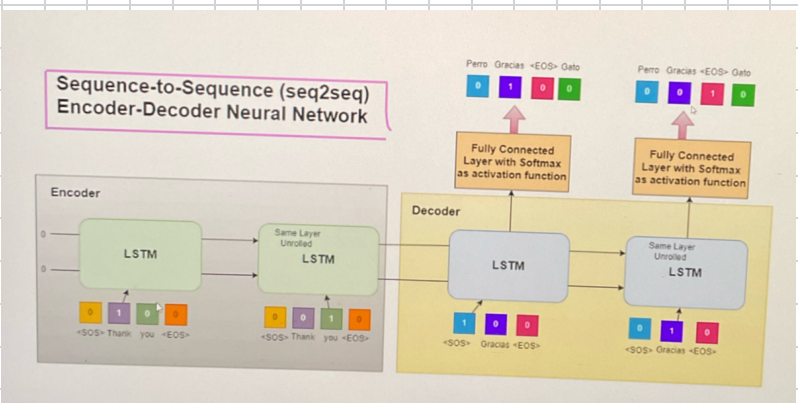
→ Context vectors are O/p of encoder.



→ This CV is passed to all the decoder units.

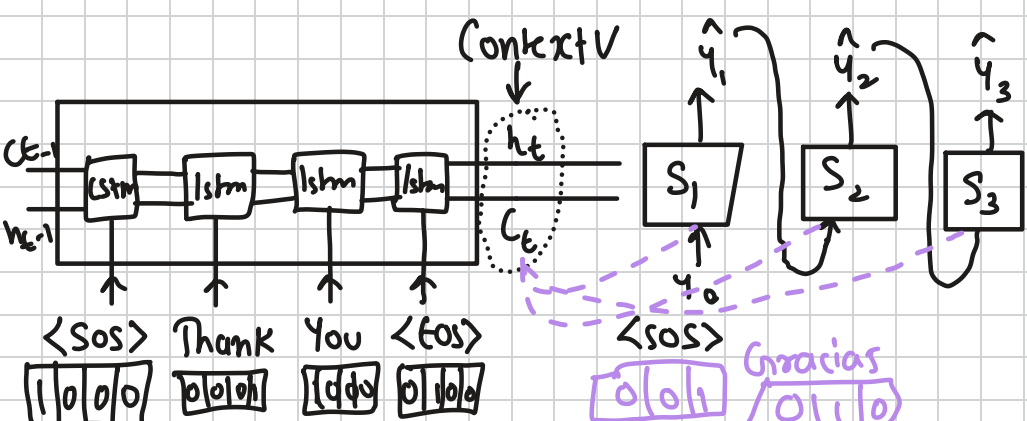
→ OP → Sequence and not words

→ Language translation, text generation



Encoder/decoder consistr of LSTM units

<SOS> Start of Sent
<EOS> End of sent
↑
Special Character



→ Context vector is a combination of C_t (cell state) & h_t (hidden)

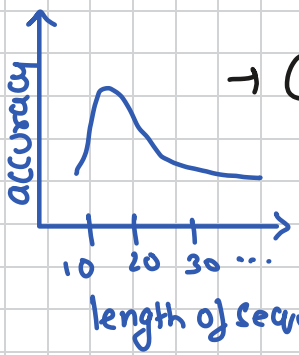
→ Encoder & decoder comprises of LSTM cells.

Context Vector W → Represents entire sentence

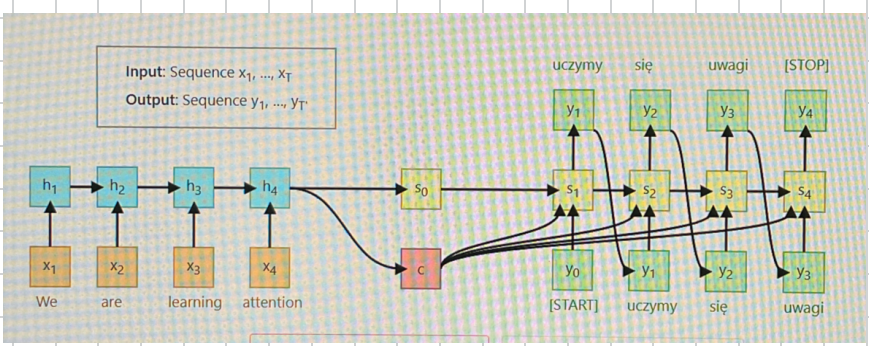
→ Bleu score → Measures model performance on length of sentence

→ for longer texts W had more context or relation with the last trained word.

→ Thatswhy Seq2seq failed for longer texts. Bleu score↓↓



→ Context Vector C and Hidden state S_0 have to summarise the whole input sequence.



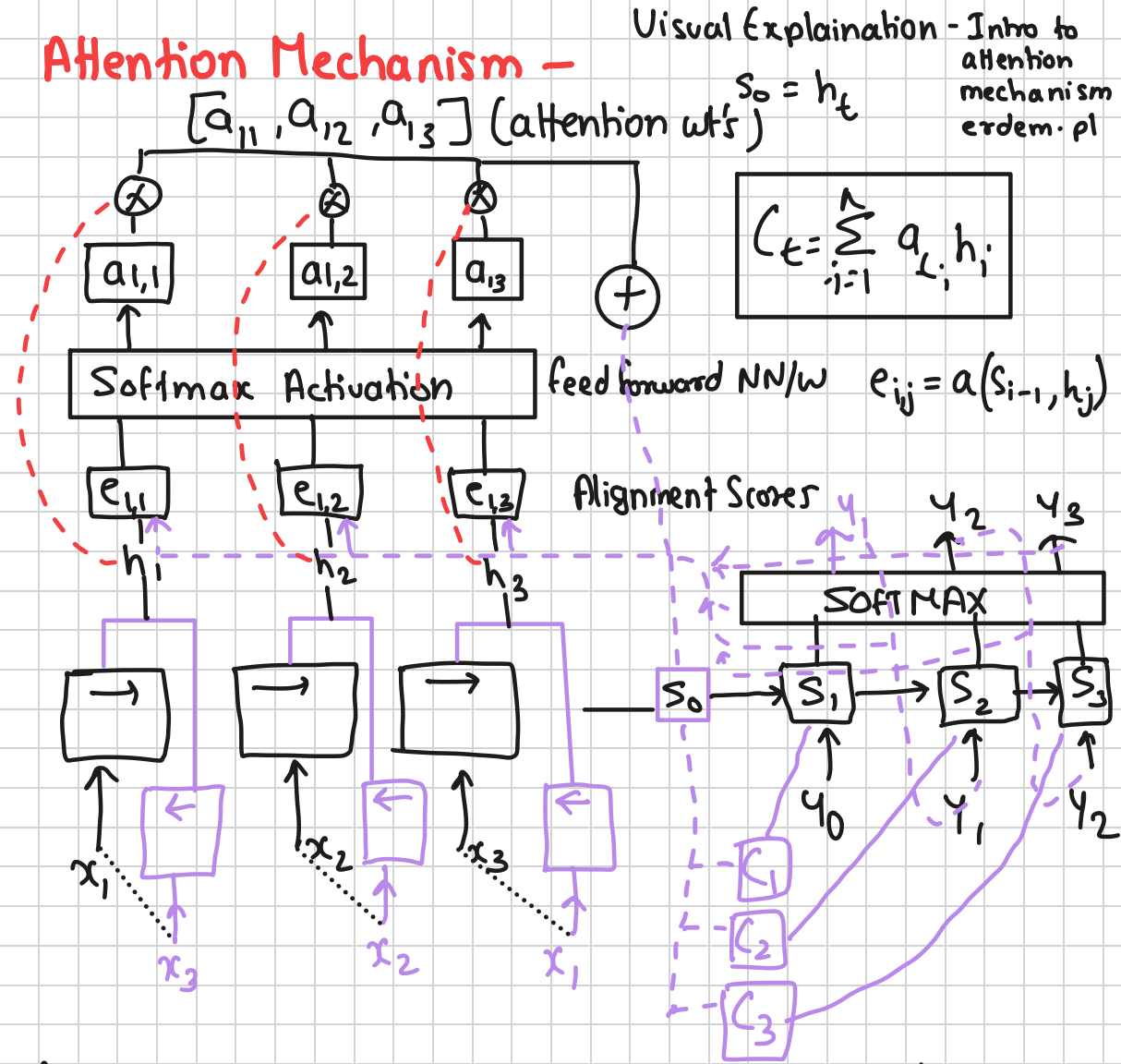
Attention Mechanism –

Visual Explaination - Intro to attention mechanism erdem.pl

$[a_{11}, a_{12}, a_{13}]$ (attention wt's)

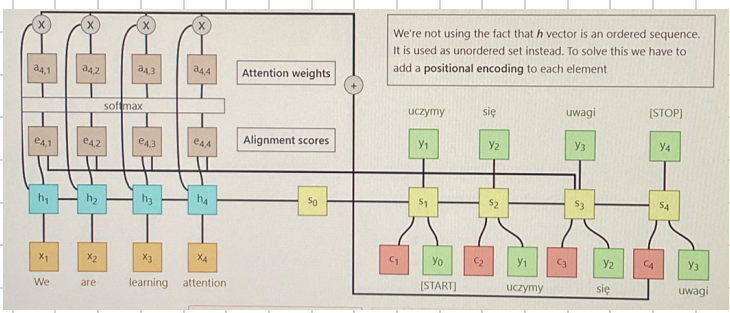
$s_0 = h_t$

$$C_t = \sum_{i=1}^{\wedge} a_{t,i} h_i$$

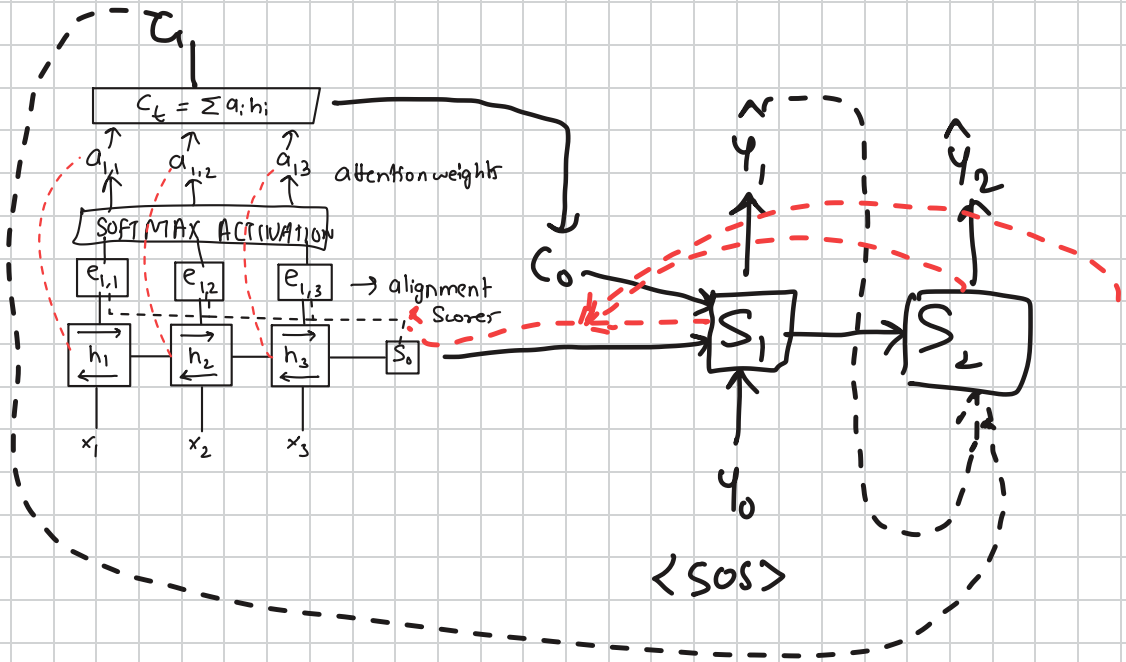


1) for each decoder output S_t a new CV → C_t is created

https://erdem.pl/2021/05/introduction-to-attention-mechanism

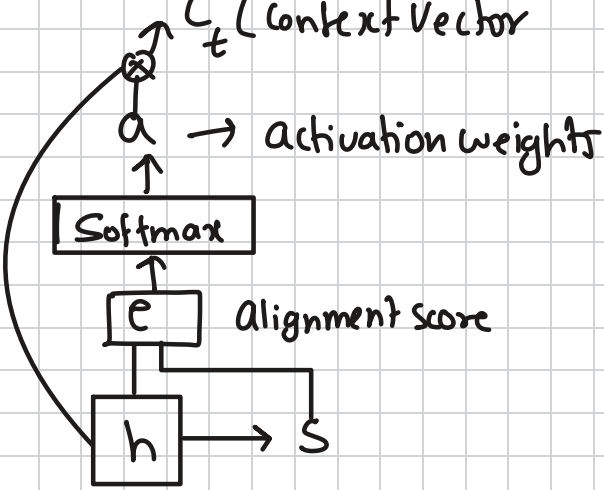


S_0 is the initial decoder state



$S_0 \rightarrow S_0 + e_{ii} \Rightarrow C_0 \rightarrow y_0 + S_0 \rightarrow S_1$

$S_1 \rightarrow S_1 + e_{ii} \Rightarrow C_1 \Rightarrow y_1 + S_1 \rightarrow S_2$



The attention mechanism in machine learning allows the model to focus on relevant parts of the input data dynamically
It works by computing, similarity score