Types of RNN Architectures

RNN Type	Flow (Input → Output)	Example	Use Case
One-to-One	Single input → Single output	Standard Feedforward NN (not really sequence-based)	Image classification (Cat vs Dog)
One-to-Many	One input → Sequence of outputs	Single image → Caption words	Image captioning, Music generation
Many-to-One	Sequence of inputs → One output	Sequence of words → Sentiment label	Sentiment analysis, Spam detection, Stock prediction
Many-to-Many (synchronous)	Sequence of inputs → Sequence of outputs (same length)	Word sequence → POS tags	Part-of-Speech tagging, Named Entity Recognition
Many-to-Many (asynchronous)	Sequence of inputs → Sequence of outputs (different length)	English sentence → French sentence	Machine translation, Speech-to-Text, Chatbots

Sentiment

Translation

Context Vector

<SOS>

(EOS)

ENCODER DECODER

Many to Many (Asynchronous)

The **Encoder–Decoder** is a **sequence-to-sequence (seq2seq)** architecture widely used in NLP tasks like **machine translation, text summarization, and chatbots.**

- **Encoder** → Compresses the input sequence into a **context vector** (a fixed-length representation of meaning).
- **Decoder** → Expands this context into an **output sequence** (possibly of different length).

Think of it as:

👉 **Encoder = "Read & Understand"**

👉 **Decoder = "Write & Generate"**

Asynchronous

Imp

CONTEXT VECTOR = is the summary of input text taken from last hidden state.

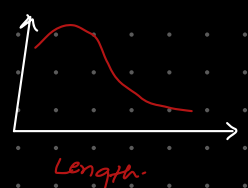
4️⃣ Step-by-Step Working (Interview-Style)

1. **Input Sequence** ($x_1, x_2, ..., x_n$) → Pass through **encoder** → produce hidden states.
2. **Final encoder hidden state** = context vector (summary of entire input).
3. **Decoder initializes with context vector.**
4. At each time step, decoder predicts the next token y_t given:
   - Previous token y_{t-1}
   - Hidden state h_{t-1}
   - Context vector (sometimes dynamically updated via attention).
5. Repeat until **<EOS> (end of sequence)** token is generated.

hidden state + previous token + context vector

Imp

DRAWBACK ;



Context vector is generally summary of text (previous states) which means

$x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x$ summary

Info Loss

To solve this → Attention Mechanism

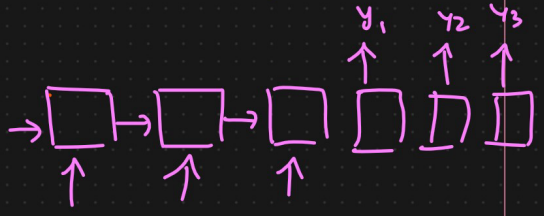
Context vector performance decrease as the text length kept on increasing.

Encoder And Decoder

1. Simple RNN → Vanishing Gradient Problem
2. LSTM RNN →
3. GRU RNN → } Long Short Term Memory.
4. Bidirectional RNN ←

Type of RNN

1. Many to Many RNN



Encoder And Decoder

Sequences I/p

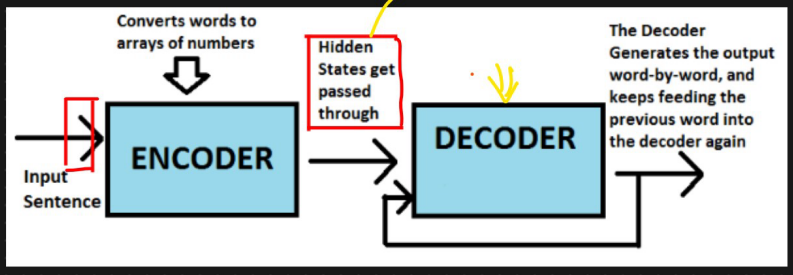
O/p Sequence Of Words

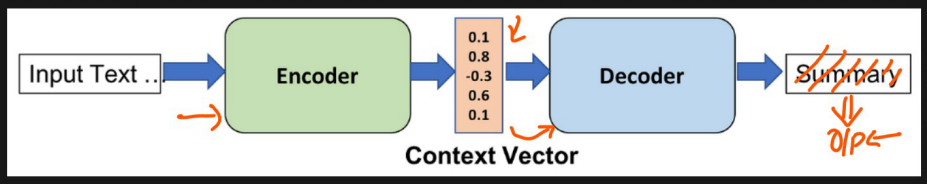
Eg: One Language To other

English → French

Eg: Linked Chat → Hi, How are you?

Simple Working





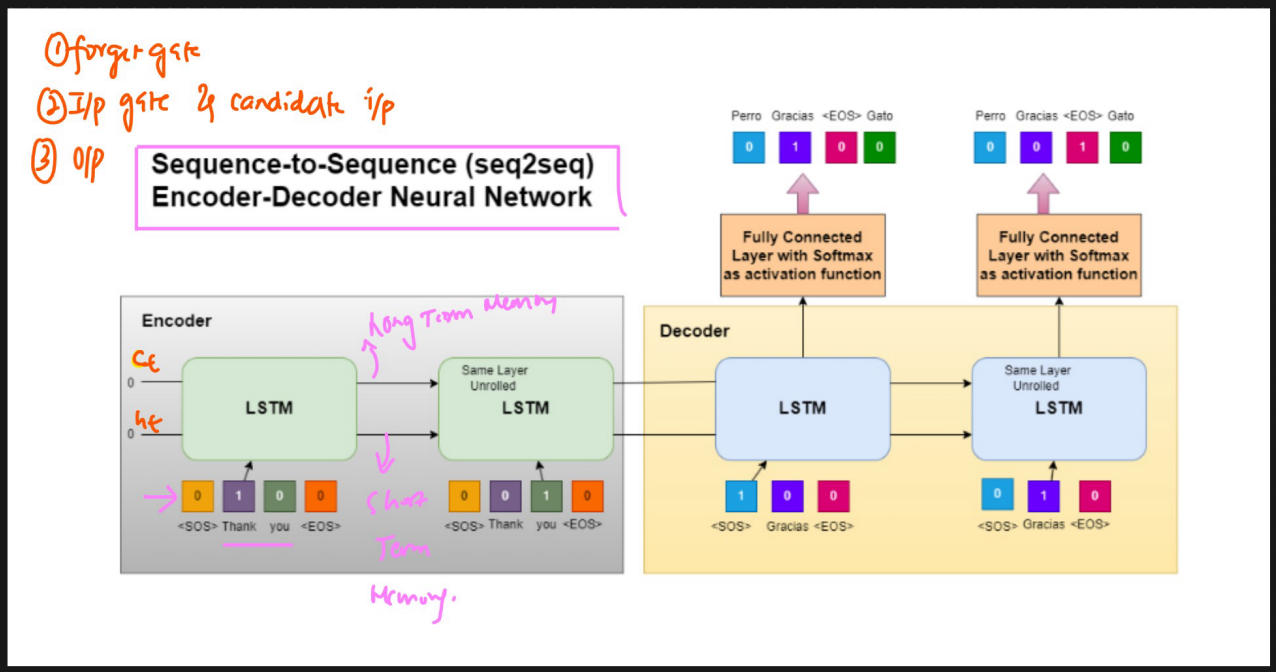
1. Encoder ⇒ I/p ⇒ Context Vector ⇐ Vectors
2. Decoder ⇒ ⇐ ⇒ O/P

Usecase

1. Language Translation
2. Text Generation
3. Text Suggestion.

RNN → Vanishing Gradient Problem

① forget gate
② I/p gate & candidate i/p
③ O/p



Dataset

English: <SOS> Thank you <EOS>. Gracias

True O/p: French {Training}: <SOS> GRACIAS <EOS>

ŷ: <EOS> . GRACIOUS <EOS>.

[y ŷ] Loss

ŷ: [0.1, 0.6, 0.3] [0.1, 0.3, 0.6] ŷ[- - -]

Encoder → Context Vectors

Long Term

Softmax Softmax

Ct, ht

Decoder

OHE ⇒ <SOS> Thank you <EOS>. Gracious

Short Term Memory

t=1 t=2 t=3 t=4

Context of this sentence.

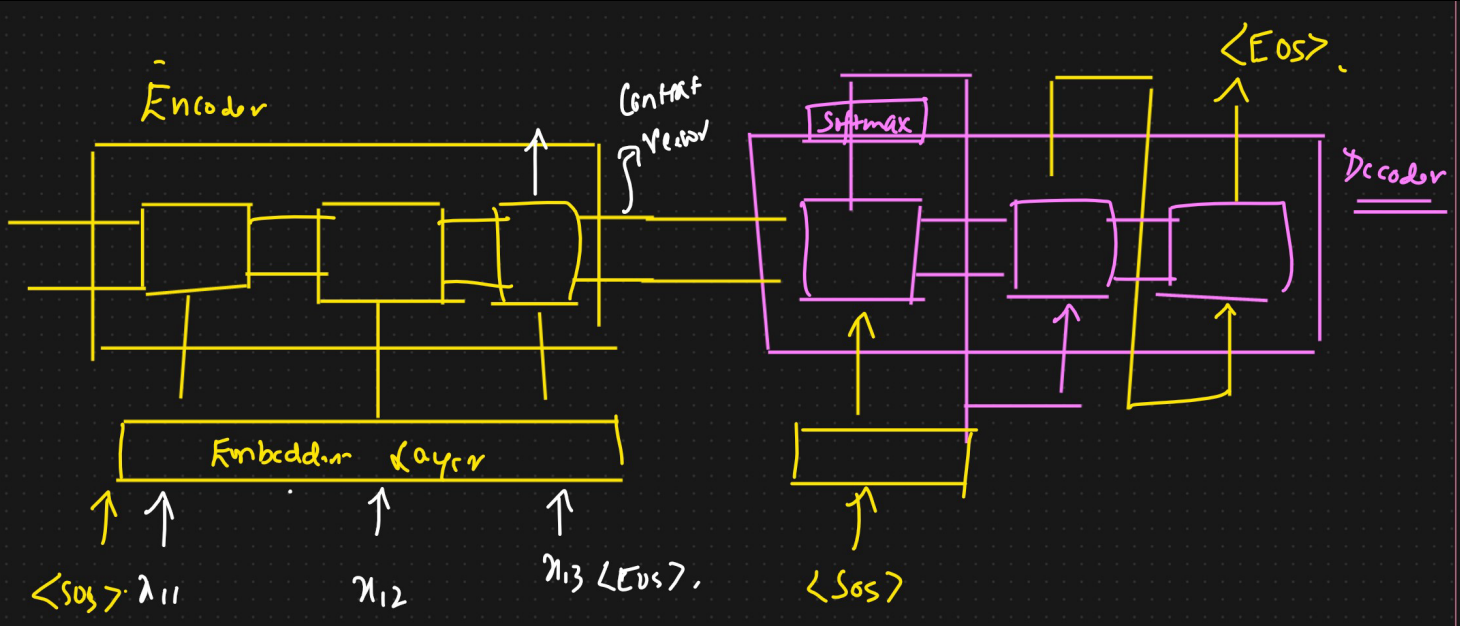
t=1 t=2 t=3

ŷ [- - -] [- - -] [- - -]

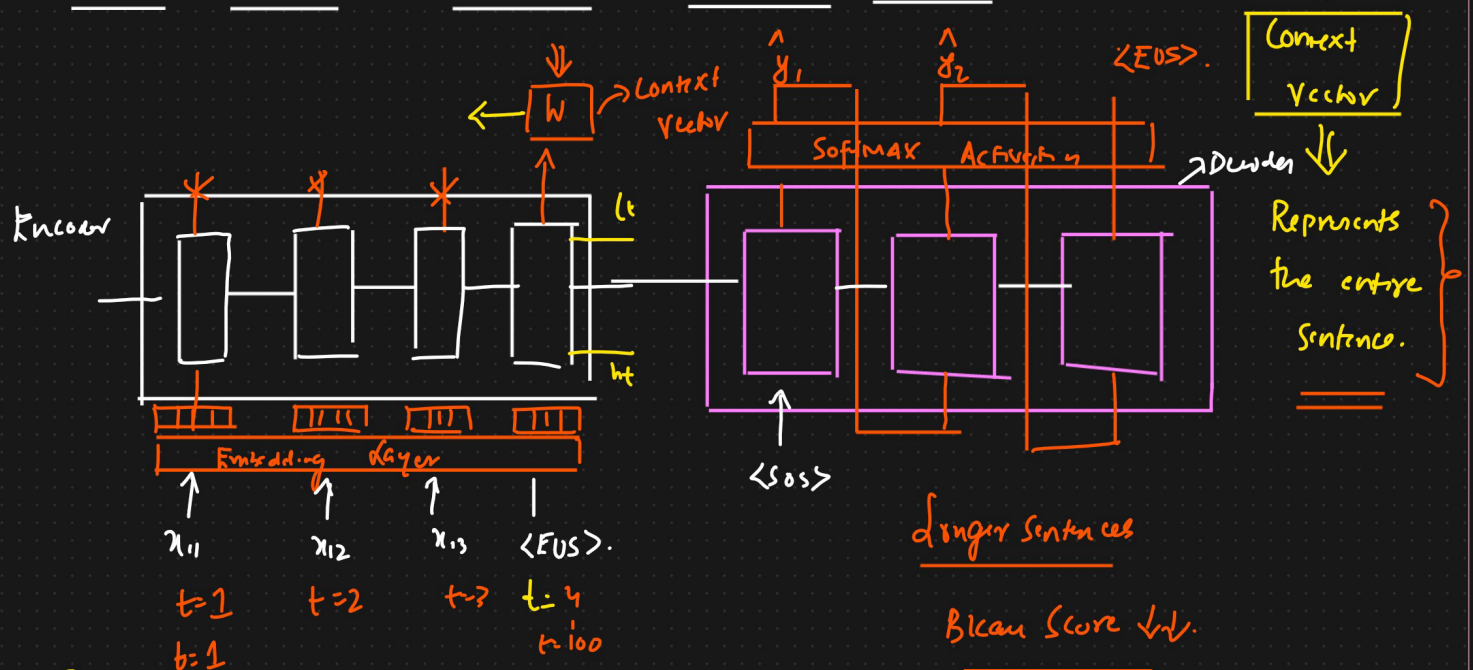
y [! ! !] [! ! !] [! ! !]

Optimizer

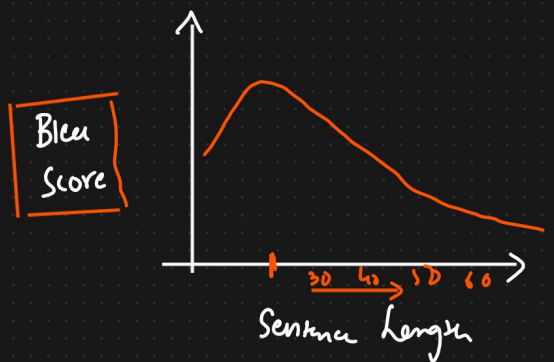
Loss $(y-\hat{y})^2$



Problems With Encoder-Decoder Seq2seq Architecture



Researchers : Sentences of varying length



⇒ Seq to Seq Data

(*) Attention Mechanism → Seq2Seq Network

Longer paragraph → {context vector}.

+

{context}