Encoder-Decoder Architecture for Sequence-to-Sequence (Seq2Seq) Models

Sequence-to-sequence (Seq2Seq) models are a cornerstone of modern Natural Language Processing (NLP) and other sequence-based tasks. They are designed to transform an input sequence into a new output sequence, where the lengths of the input and output sequences can differ. This makes them incredibly versatile for a wide range of applications. The core of the Seq2Seq framework is the **Encoder-Decoder architecture**.

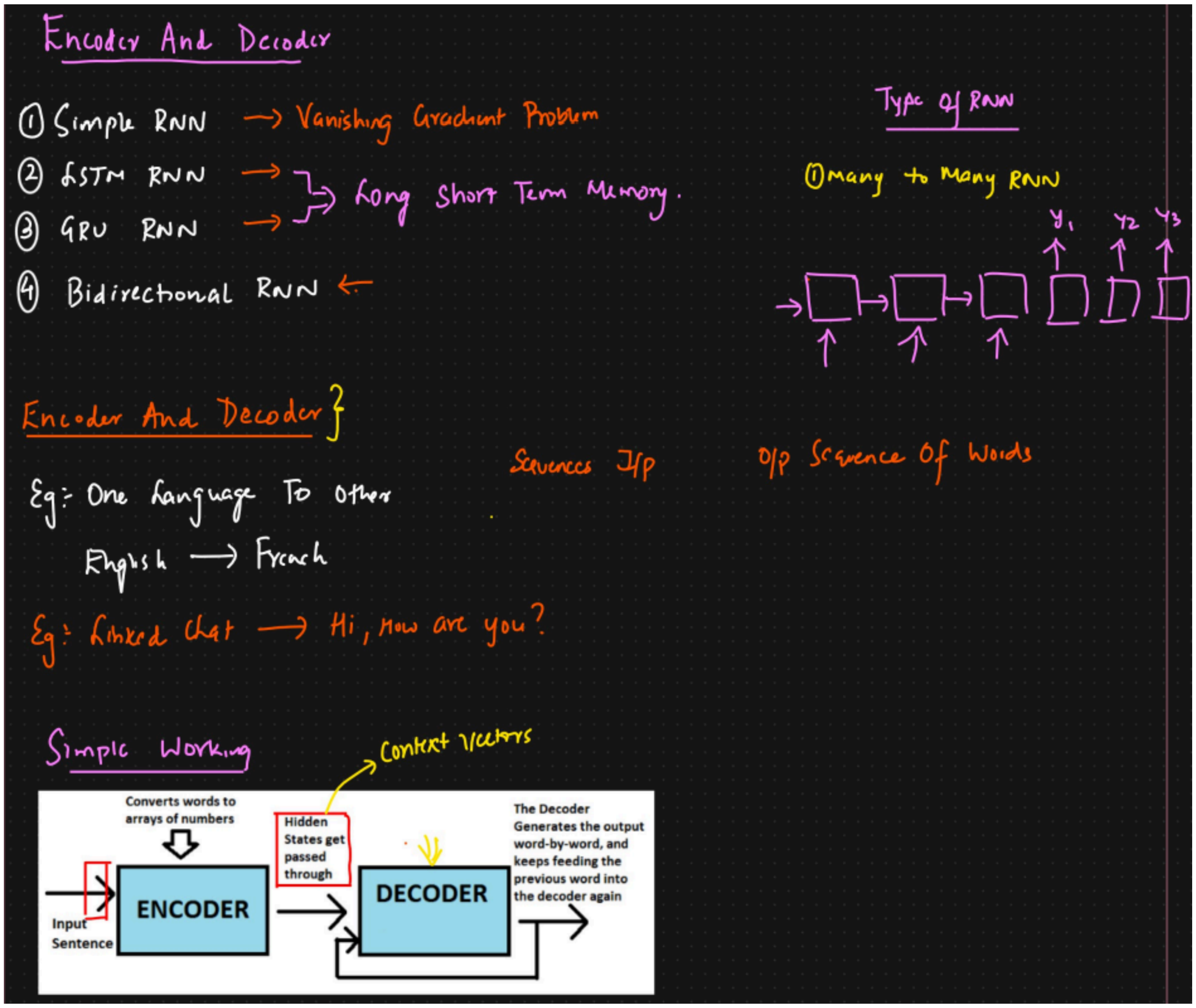
1. The Core Idea: Overcoming Fixed-Length Vectors

Before Seq2Seq, traditional neural networks struggled with tasks where the input and output sequences had variable lengths. For example, in machine translation, the number of words in a source sentence is rarely the same as in the target translation.

The Encoder-Decoder architecture solves this by using two main components:

- **An Encoder:** This part reads the entire input sequence and compresses it into a single, fixed-length vector that encapsulates its meaning. This vector is often called the **context vector** or "thought vector."
- **A Decoder:** This part takes the context vector from the encoder and generates the output sequence one element at a time.

This design decouples the input and output sequences, allowing for flexible mapping between sequences of different lengths.



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

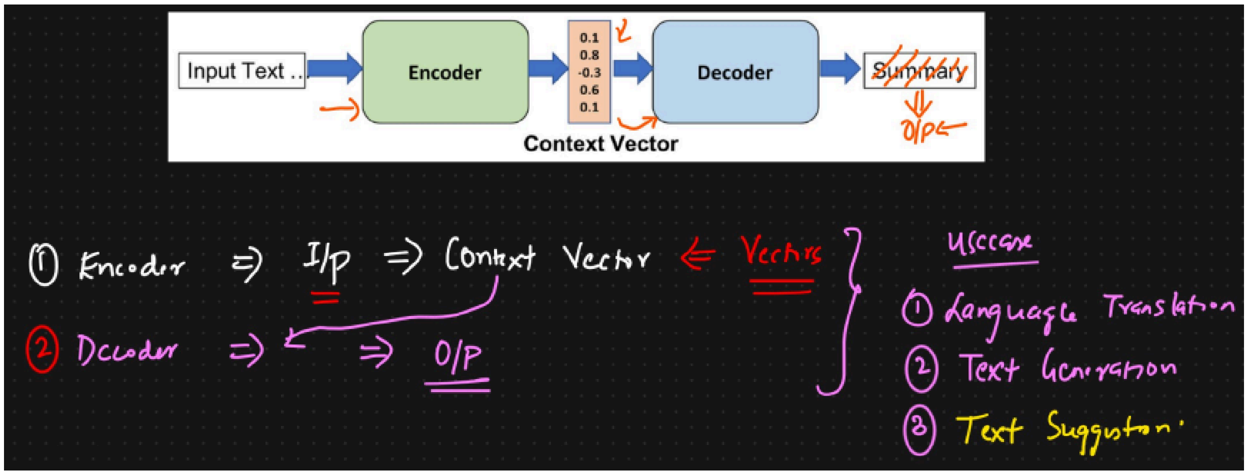
Eg: One Language To other

English → French

Eg: Linked Chat → Hi, How are you?

Simple Working

y1
y2
y3
Context Vectors
Converts words to arrays of numbers
Hidden States get passed through
The Decoder Generates the output word-by-word, and keeps feeding the previous word into the decoder again
ENCODER
DECODER
Input Sentence



① Encoder ⇒ I/p ⇒ Context Vector ⇐ Vectors

② Decoder ⇒ ⇒ O/P

Usecase

① Language Translation

② Text Generation

③ Text Suggestion.

2. The Architecture in Detail

Typically, both the encoder and the decoder are implemented using Recurrent Neural Networks (RNNs), most commonly **Long Short-Term Memory (LSTM)** or Gated Recurrent Unit (GRU) cells, due to their ability to handle long-range dependencies.

The Encoder

The encoder is an RNN that processes the input sequence token by token (e.g., word by word).

- **Input:** The input sequence is fed into the encoder one token at a time. For instance, the sentence "How are you" would be input as three separate tokens.
- **Processing:** At each time step, the RNN updates its hidden state based on the current input token and the previous hidden state. The hidden state at each step captures information from the preceding tokens.
- **Output (The Context Vector):** After processing the entire input sequence, the final hidden state of the encoder RNN is considered the **context vector**. This vector is a numerical summary of the entire input sequence's information. The outputs at each time step of the encoder are usually discarded; only the final state is passed to the decoder.

Let the input sequence be X=(x1,x2,...,xn). The encoder's hidden state at time step t is calculated as:

$h_t = f(W_h h_{t-1} + W_x x_t)$The context vector, C, is the final hidden state:

$$C = h_n$$

The Decoder

The decoder is also an RNN that generates the output sequence token by token.

- **Initialization:** The decoder is initialized with the context vector from the encoder. This context vector is used as the initial hidden state of the decoder RNN. This "conditions" the decoder on the meaning of the input sequence.
- **Generation Process (Autoregressive):**
    1. The decoder starts with a special "start-of-sequence" (`<SOS>`) token as its first input.
    2. At each time step, the decoder takes its previous hidden state and the previously generated token as input to produce the next token in the output sequence.
    3. This process continues until a special "end-of-sequence" (`<EOS>`) token is generated, signaling that the output is complete.
- **Output:** The output of each decoder time step is a probability distribution over the entire vocabulary (e.g., all possible words in the target language). The token with the highest probability is chosen as the output for that step.

The decoder's hidden state at time step t is:

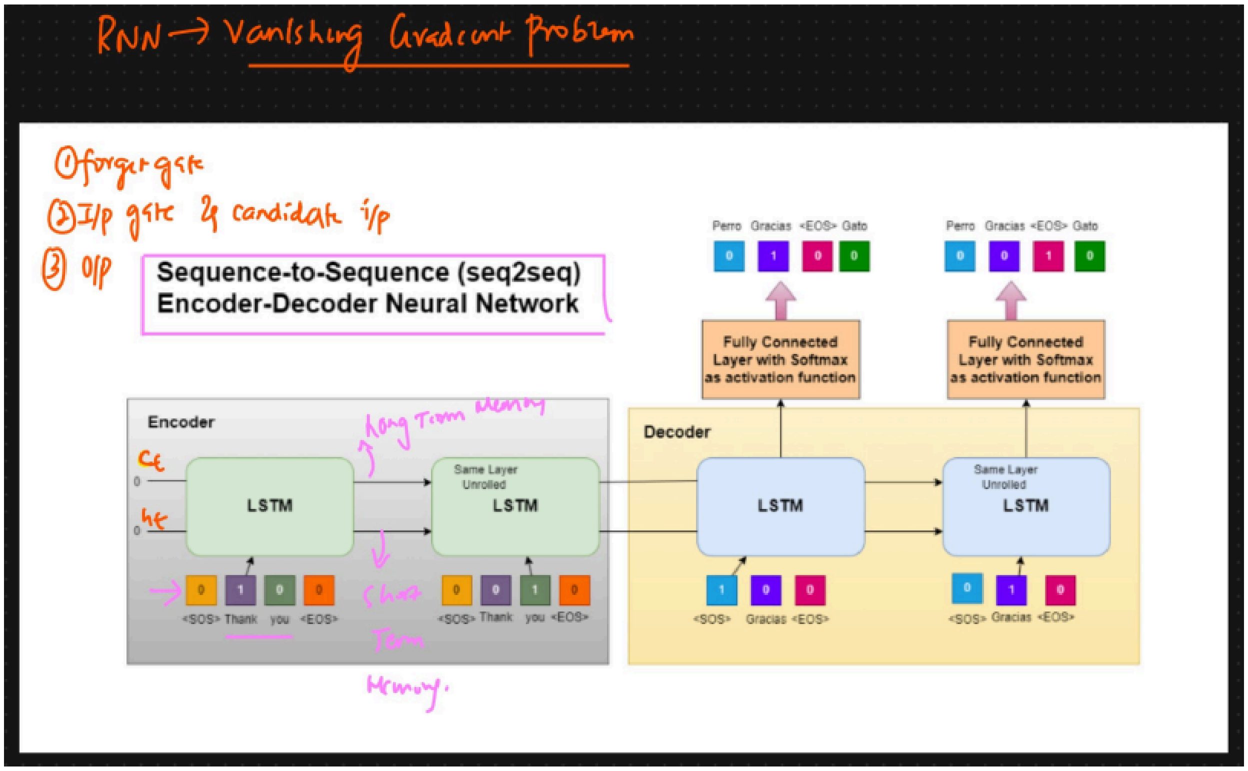
$$ \$s_t = g(W_s s_{t-1} + W_y y_{t-1}) Thepredictedoutputtokenis : \hat{y}_t = \mathrm{softmax}(W_{out} s_t)\$ $$

where yt−1 is the token generated at the previous step, and the initial hidden state s0 is the context vector C.

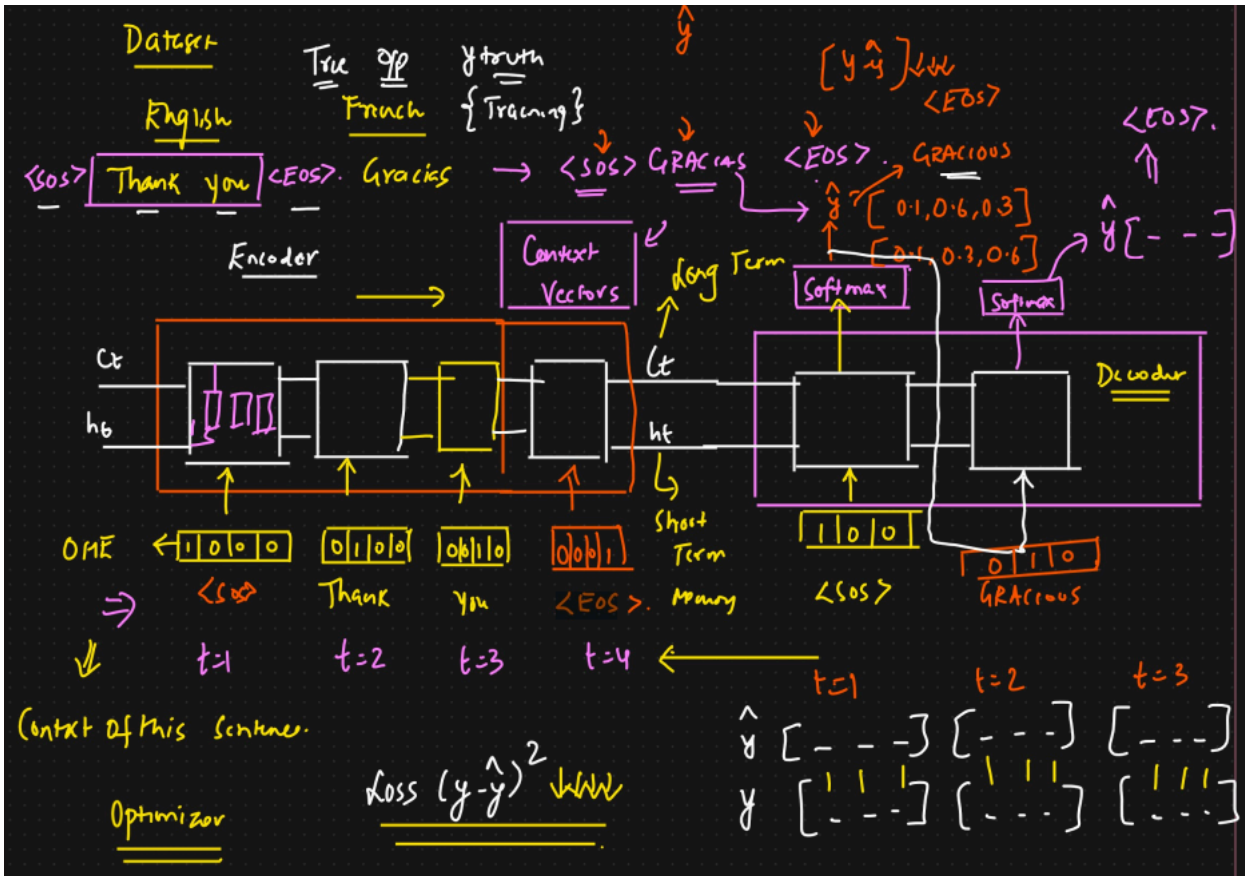
3. Training the Model: Teacher Forcing

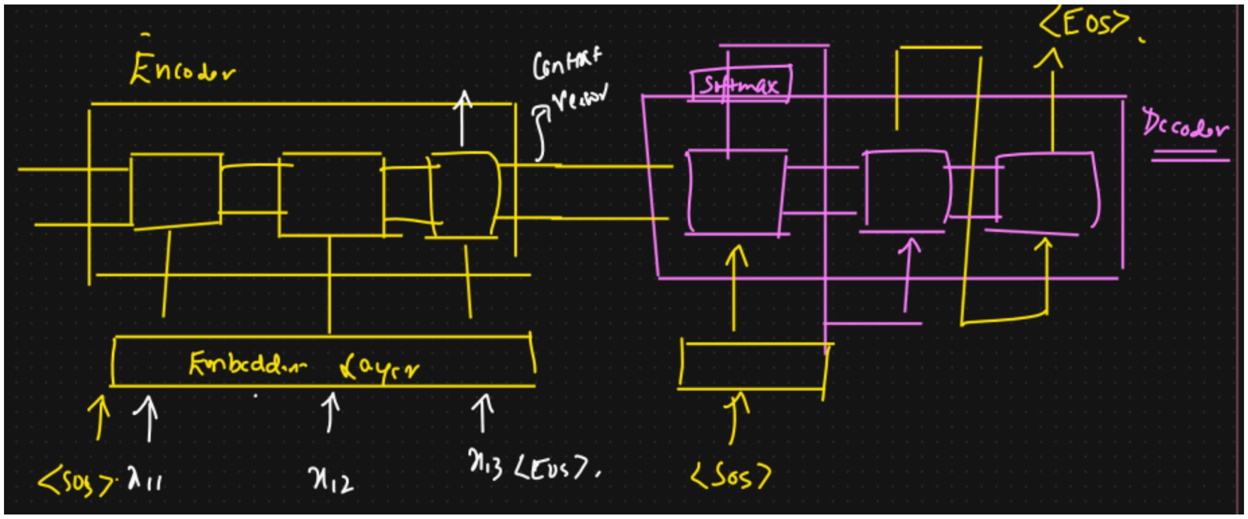
Seq2Seq models are trained to maximize the probability of the correct output sequence given the input sequence. A common and effective technique used during training is **Teacher Forcing**.

- **How it Works:** Instead of feeding the decoder's own (potentially incorrect) prediction from the previous time step as input for the current time step, we feed the **actual** ground-truth token from the target sequence.
- **Advantages:** This method significantly stabilizes the training process and helps the model converge much faster. It prevents the propagation of errors during training, as the decoder is always guided by the correct sequence.
- **Disadvantage (Exposure Bias):** During inference (when making predictions on new data), there are no ground-truth outputs to guide the decoder. The model is now "exposed" to its own predictions, which might be imperfect. This discrepancy between training and inference can lead to a degradation in performance.



RNN → Vanishing Gradient Problem
① forget gate
② I/p gate & candidate i/p
③ O/p
Sequence-to-Sequence (seq2seq) Encoder-Decoder Neural Network
Encoder
Ct
ht
LSTM
Same Layer Unrolled
<SOS> Thank you <EOS>
Long Term Memory
Short Term Memory.
Decoder
<SOS> Gracias <EOS>
Fully Connected Layer with Softmax as activation function
Perro Gracias <EOS> Gato





4. Applications of Seq2Seq Models

The versatility of the Encoder-Decoder architecture has led to its successful application in numerous fields:

- **Machine Translation:** The canonical example, translating a sentence from a source language to a target language (e.g., Google Translate).

- **Text Summarization:** Taking a long document as input and generating a short, coherent summary.
- **Chatbots and Conversational AI:** Generating a response based on a user's query or statement.
- **Image Captioning:** Taking an image (processed by a CNN to produce a feature vector, which acts as the context) and generating a descriptive sentence.
- **Speech Recognition:** Converting an audio waveform (a sequence of audio features) into a sequence of text.
- **Code Generation:** Translating natural language descriptions into computer code.

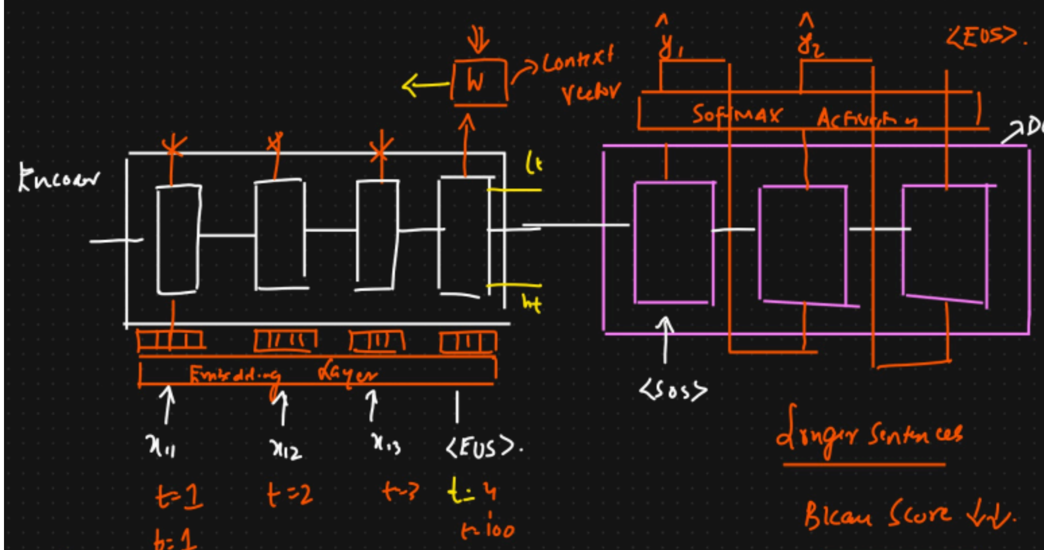
5. Limitations and Challenges

The basic Encoder-Decoder architecture, while powerful, has a key limitation:

- **The Information Bottleneck:** The encoder must compress all the information from the input sequence into a single, fixed-length context vector. For long sequences, this can lead to a loss of crucial information, as the vector struggles to retain details from the beginning of the sequence. This makes the model's performance degrade significantly as the input length increases.

This limitation was a major motivation for the development of the **Attention Mechanism**, which is now a standard addition to most Seq2Seq models. Attention allows the decoder to look back at the entire sequence of the encoder's hidden states at each step of the output generation, effectively overcoming the information bottleneck.

Problems With Encoder-Decoder Seq2seq Architecture



Context Vector

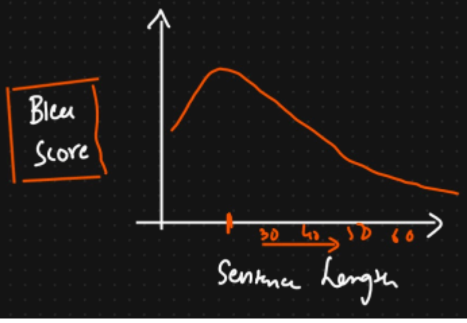
⇓

Represents the entire sentence.

Longer Sentences

Bleau Score ↓↓.

Researchers : Sentences of varying length



⇒ Seq to Seq Data

⊛ Attention Mechanism ⟶ Seq2Seq Network

Longer paragraph ⟶ {Context Vector}.

+

{Context}