**Encoder-Decoder**

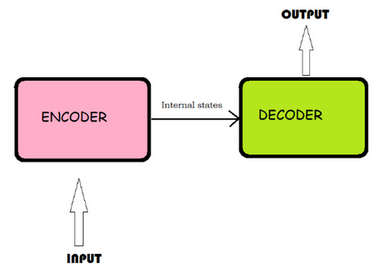
======================================================================

**Pre-requisite:** Seq2Seq model – which takes sequence of input values (example: sentences or paragraphs) and produces the output sequences (example: sentences or paragraphs)

**Ex:** Language translation, Summarization, chatbot responses, Q & A’s.

--------------------------------------------------------------------------------------------------------------

Encoder-Decoder architecture is a **sequence-to-sequence (Seq2Seq**) model structure, originally designed for tasks like machine translation but can used in many areas of NLP, speech, and even vision.

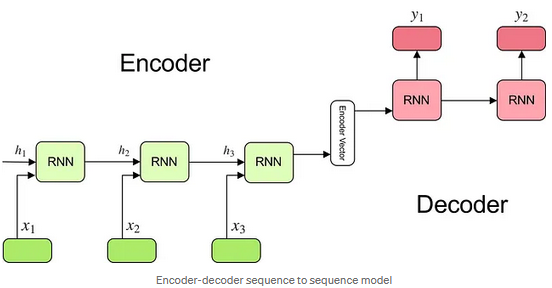


--------------------------------------------------------------------------------------------------------------

**What is Encoder-Decoder Architecture?**

It’s a two-part neural network design used to transform one sequence into another.

* **Encoder**: Processes the input sequence and encodes it into a fixed-length context vector.
* **Decoder**: Takes the context vector and generates the output sequence step-by-step.



1. **Encoder**

* Typically, an RNN/LSTM/GRU (or Transformer encoder)
* It takes the input sequence (e.g., a sentence in English) and processes it one timestep at a time.
* It encodes the full sequence into a single vector (often the final hidden state), known as the context vector.

2. **Context Vector**

* A compressed representation of the input sequence.
* Passed to the decoder.
* This bottleneck vector is supposed to contain all the relevant information needed for output generation.

3. **Decoder**

* Another RNN/LSTM/GRU (or Transformer decoder)
* It takes the context vector and generates the output sequence (e.g., a sentence in French) token by token.
* The decoder uses previous outputs as inputs for the next prediction step (auto-regressive generation).

**Advantages**

* Can handle variable-length input and output sequences.
* Works well for translation, summarization, chatbot responses, etc.
* Flexible: You can replace RNNs with LSTM/GRU/Transformer blocks.

-------------------------------------------------------------------------------------------------------------

**Drawbacks**

1. Fixed-size context vector bottleneck:
   * If the input is long (e.g., a paragraph), compressing all info into one vector hurts performance.
   * Solution: Use attention mechanism (which led to Transformers).
2. Slow inference:
   * Decoding is sequential (word-by-word); can’t parallelize easily.
   * Transformers solve this partially.
3. Memory constraints:
   * Long sequences are harder to model for basic RNN/LSTM-based architectures.

--------------------------------------------------------------------------------------------------------------

**Modern Improvement: Attention + Transformer**

* Attention lets the decoder look at all encoder hidden states, not just a single context vector.
* Transformer (like BERT, GPT) uses self-attention instead of RNNs altogether for more parallelism and context awareness.