**Part-I: Theoretical Understanding of RNN, LSTM, and Encoder-Decoder**

**Objective:**

To understand the architecture and working of Recurrent Neural Networks (RNN), Long Short-Term Memory (LSTM), and the encoder-decoder structure for sequence-to-sequence tasks.

**Assignment Tasks**

**Task 1: Conceptual Questions**

Answer the following questions in 2–4 sentences each:

1. What is the difference between RNN and LSTM?

Ans- RNNs try to remember things step-by-step as they go through a sequence, but their memory fades with longer inputs. LSTMs are like RNNs with a better memory system—they have gates that decide what to remember, forget, or pass on, making them way more reliable for long texts or speech.

2. What is the vanishing gradient problem, and how does LSTM solve it?

Ans- When training normal RNNs, the learning signal can shrink so much that it stops reaching earlier layers—like a message passed in a long line of people that gets lost. LSTM uses a clever “cell state” and gates that help keep the signal strong, so it doesn’t lose important info from earlier parts of the sequence.

3. Explain the purpose of the Encoder-Decoder architecture.

Ans- This setup is like a tag team: the encoder reads and understands the whole input, like a translator listening to a full sentence in English. The decoder then uses that understanding to create a new sentence, maybe in Hindi or French, step by step.

4. In a sequence-to-sequence model, what are the roles of the encoder and decoder?

Ans- The encoder acts like the “listener” that understands the input and packs the meaning. The decoder is the “speaker” who uses that packed info to generate a new output, one word or step at a time, based on what it's learned and remembered.

5. How is attention different from a basic encoder-decoder model?

Ans- In basic models, the decoder only gets one summary of the input—like reading a book and only using your memory to explain it. Attention lets the decoder “peek” at different parts of the original input whenever it needs to, making responses smarter and more accurate.

**Task 2: Sequence-to-Sequence Data Flow**

Draw or describe the **data flow** in an encoder-decoder model using RNN/LSTM. Clearly label:

● Input sequence

● Hidden states

● Context vector

● Output sequence

In a sequence-to-sequence model with RNN or LSTM, the data flows through two main components: the **encoder** and the **decoder**.

Suppose the input is: **"How are you today?"**  
The goal is to generate a response like: **"I'm doing great!"**

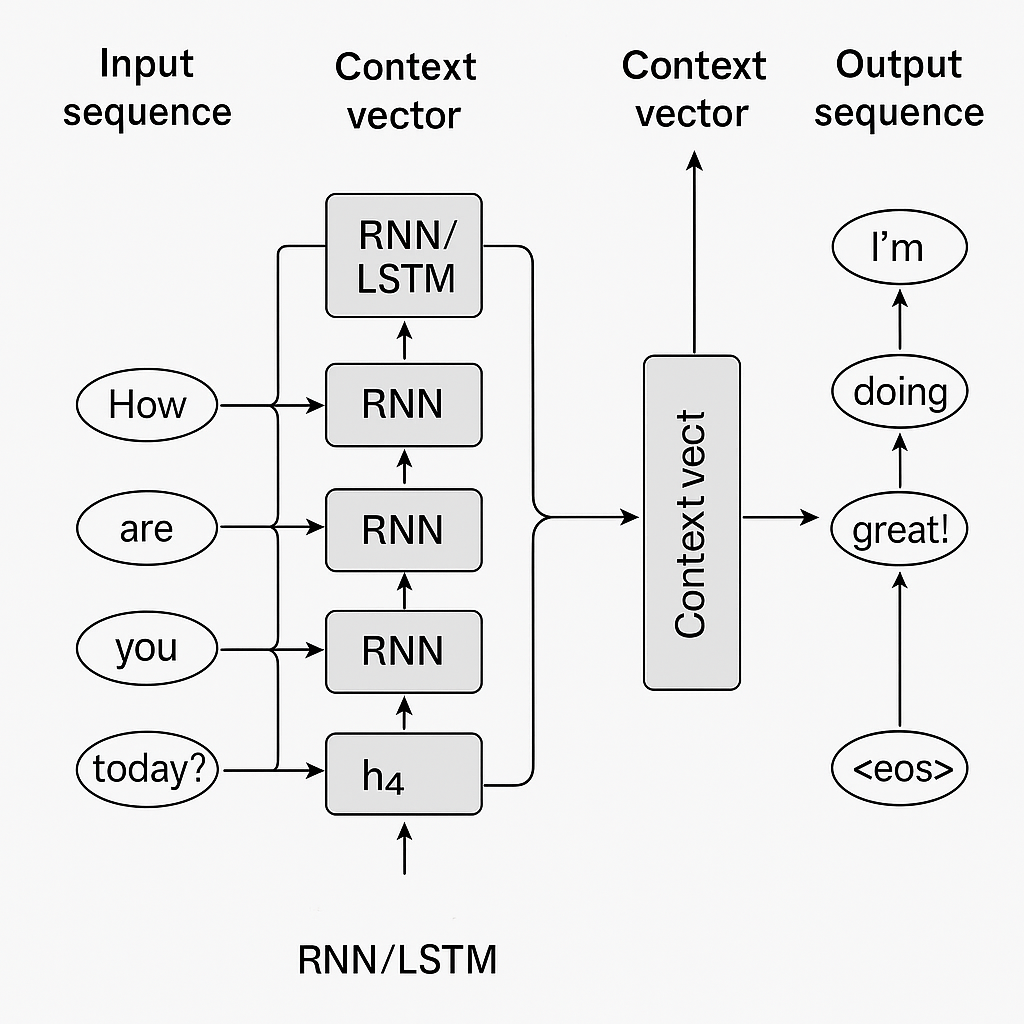
**Input Sequence**:  
The input sentence is tokenized into words (“How”, “are”, “you”, “today?”), and each is fed one by one into the encoder.

**Encoder Hidden States**:  
As each word enters, the encoder updates its hidden states. The final hidden state captures the overall meaning of the sentence.

**Context Vector**:  
This final hidden state (context vector) summarizes the input and is passed to the decoder.

**Decoder Hidden States**:  
The decoder uses the context vector and begins generating the output word by word, using its current hidden state and the previously generated word.

**Output Sequence**:  
Words like “I’m”, “doing”, and “great!” are generated until the model predicts the end of the sentence.



**Task 8: Model Performance Discussion**

**1. What are the challenges in training sequence-to-sequence models?**

Training sequence-to-sequence models isn’t always smooth. Here are some common struggles:

* **Forgetting earlier words**: If the input sentence is long, the model might forget the beginning while it's still processing the end.
* **Not enough good examples**: The model learns from examples. If the training data is small or low quality, it won’t perform well.
* **Hard to handle rare words**: If a word doesn’t appear often in training, the model may not know what to do with it.
* **Takes a lot of time and power**: Training these models requires strong computers and plenty of time.
* **Confusion during prediction**: While training, the model sees correct answers. But during testing, it must guess on its own, which can lead to mistakes.

**2. What does a “bad” translation look like? Why might it happen?**

A bad translation is one that sounds awkward, wrong, or loses the original meaning.

**Example**:  
Original: *"He went to the bank to deposit money."*  
Bad translation: *"He went to the river bank to put the money."*

**Why does this happen?**

* The model misunderstood the word "bank" (which can mean a place for money or a riverbank).
* It didn’t have enough context to choose the right meaning.
* It may not have seen enough similar examples while training.
* It could also mess up the sentence structure if it’s translating between very different languages.

**3. How can the model be improved further?**

There are several ways to make these models better:

* **Help the model focus**: Using attention mechanisms allows it to concentrate on the most important words while generating the output.
* **Give it more and better data**: The more examples it sees, the better it learns.
* **Break words into smaller parts**: This helps it handle uncommon or new words.
* **Fine-tune on specific topics**: First train it generally, then specialize it (e.g., medical or legal language).
* **Add more context**: Let the model see previous sentences or extra info to understand better what’s going on.