# Task 1: Conceptual Questions

1. What is the difference between RNN and LSTM?

RNNs (Recurrent Neural Networks) are designed to process sequential data by maintaining a hidden state that captures information from previous steps in the sequence. However, basic RNNs struggle with long-term dependencies due to the vanishing or exploding gradient problem. LSTMs (Long Short-Term Memory networks) are a type of RNN specifically designed to overcome these limitations by incorporating 'gates' (input, forget, and output gates) that regulate the flow of information into and out of the cell state, allowing them to learn and remember long-term dependencies more effectively.

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

The vanishing gradient problem occurs during the training of RNNs when the gradients, which carry information used to update network weights, shrink exponentially as they are propagated backward through many layers or time steps. This makes it difficult for the network to learn long-term dependencies. LSTMs solve this by using a cell state and various gates (input, forget, output gates) that control the flow of information. The forget gate, for instance, decides what information to discard, while the input gate decides what new information to store, allowing the gradient to flow more directly through the cell state without vanishing.

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

The Encoder-Decoder architecture is a neural network design primarily used for sequence-to-sequence tasks, where the input and output are both sequences but often of different lengths and complexities. Its purpose is to transform an input sequence into a meaningful fixed-length 'context vector' (the encoder's role) and then to generate an output sequence from that context vector (the decoder's role). This separation allows the model to handle variable-length inputs and outputs effectively.

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

The encoder reads the entire input sequence, processes it step by step, and compresses all relevant information into a fixed-size context vector. The decoder then takes this context vector and generates the output sequence, one element at a time.

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

In a basic encoder-decoder model, the decoder relies solely on a single fixed-size context vector generated by the encoder, which can be a bottleneck for long sequences. Attention mechanisms allow the decoder to selectively focus on different parts of the input sequence at each output step by computing a weighted sum of all encoder hidden states, leading to better performance.

# Task 2: Sequence-to-Sequence Data Flow

The data flow in an encoder-decoder model using RNN/LSTM is as follows:

* Input sequence: The input sequence is fed into the encoder one element at a time.
* Hidden states (Encoder): The encoder updates its internal hidden state at each step.
* Context vector: The final hidden state of the encoder is used as the context vector.
* Hidden states (Decoder): The decoder initializes with the context vector.
* Output sequence: The decoder generates output elements sequentially.

# Part III: Visualizing and Enhancing Encoder-Decoder

Objective: To interpret and improve encoder-decoder outputs with attention and performance visualization.

## Task 7: Plotting Loss and Accuracy

Observations on Overfitting:

* Training loss decreases while validation loss increases — model is memorizing training data.

Observations on Underfitting:

* Both losses are high and accuracy is low — model lacks capacity or training.

Observations on Training Stability:

* Smooth decrease in loss and increase in accuracy — training is stable.

## Task 8: Model Performance Discussion

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

* Long-term dependencies
* Variable sequence lengths
* Exposure bias
* Evaluation metrics like BLEU may not capture fluency
* High computational cost

2. What does a 'bad' translation look like? Why might it happen?

* Inaccuracies or mistranslations due to misunderstanding context
* Grammatical errors or awkward phrasing
* Omissions or unnecessary additions
* Repetitions and loops
* Inconsistent tone or style

3. How can the model be improved further?

* Implement attention mechanism
* Use bi-directional encoder
* Train on larger or augmented dataset
* Tune hyperparameters
* Use beam search decoding
* Apply regularization
* Use pre-trained embeddings
* Gradually reduce teacher forcing ratio
* Explore Transformer-based models