# Assginment 2

## Encoder-Decoder Models using RNN and LSTM

## Task 1

Q 1 - What is the difference between RNN and LSTM?

Ans - RNNs remember past info but forget quickly.
LSTMs are better at holding onto important info for longer.
So, LSTMs handle longer sequences more accurately than RNNs.

Q2 - What is the vanishing gradient problem, and how does LSTM solve it?

Ans - The vanishing gradient problem happens when gradients become too small during backpropagation, making it hard for neural networks to learn long-term dependencies. LSTMs solve this by using special gates and a cell state that let them preserve and control the flow of information over time.

Q3 - Explain the purpose of the Encoder-Decoder architecture?

Ans - The Encoder-Decoder architecture is used to convert one sequence into another, like translating a sentence from English to French. The encoder compresses the input into a fixed-size context, and the decoder uses that to generate the output. It's like understanding a message and then rephrasing it in another language.

Q4 - In a sequence-to-sequence model, what are the roles of the encoder and decoder?

Ans - In a sequence-to-sequence model, the encoder reads the input sequence and turns it into a context vector that captures its meaning. The decoder takes that context and generates the output sequence, step by step. Think of the encoder as listening and understanding, and the decoder as responding or translating.

Q5- How is attention different from a basic encoder-decoder model?

Ans - In a basic encoder-decoder model, the decoder relies only on a single fixed context vector from the encoder. Attention lets the decoder look at different parts of the input sequence at each step, focusing on what matters most. It's like having a spotlight that shifts to the most relevant words while translating or generating.

## Task 2

Q1- Draw or describe the data flow in an encoder-decoder model using RNN/LSTM. Clearly label:

● Input sequence

● Hidden states

● Context vector

● Output sequence

Ans - Input sequence (like a sentence) goes into the encoder, one word at a time.

The encoder builds up hidden states, and the final one becomes the context vector — a summary of the whole input.

The decoder uses this context to generate the output sequence, word by word.

## Implementation of Encoder-Decoder using RNN / LSTM

## Task 3

Q1- Use a toy dataset or download an English-to-French translation dataset (e.g., from http://www.manythings.org/anki/). ● Preprocess the text: ○ Tokenize input and output sequences ○ Pad sequences for batching ○ Prepare input_tensor, target_tensor?

Ans - 1. Load the dataset

"I am happy." → "Je suis heureux."

2. Tokenize input and output

Break sentences into words and map them to numbers (word → index):

"I am happy" → [12, 45, 78]

3. Pad sequences

input_tensor: tokenized and padded English sentences

target_tensor: tokenized and padded French sentences

## Task 4

Q1- Define an encoder model using Embedding + LSTM ?

Ans - An encoder model uses an Embedding layer to turn words into vector form the model can understand. Then, an LSTM layer reads the sequence and summarizes it into hidden states. These hidden states become the context passed to the decoder for translation or generation.

Q2- Define a decoder model using Embedding + LSTM + Dense ?

Ans- A decoder model uses an Embedding layer to convert target words into vectors. Then, an LSTM layer processes these vectors, guided by the encoder's context states. Finally, a Dense layer predicts the next word in the output sequence, step by step.

It's like turning ideas into words, one at a time!

Q3 - Compile and train the model for at least 10 epochs ?

Ans - To compile and train the model, you set up the optimizer and loss function, then run the training for 10 epochs.

For example, use Adam optimizer and sparse categorical cross-entropy loss, then call model.fit() with your data.

It's like tuning your model by showing it examples repeatedly so it gets better over time!

Q4- Print the training loss after each epoch.

Ans - **"Epoch 1 finished, training loss is 0.45 — the model is learning!"**

**"Epoch 2 done, training loss dropped to 0.30 — getting better!"**

## Task 5

Q1 - Implement inference (testing) using a loop:

a. Feed encoder with input sequence

b. Predict one word at a time from the decoder using teacher forcing or greedy search

Ans - For inference, feed the input sequence into the encoder to get the context. Then, start the decoder with a start token and predict one word at a time, using greedy search (picking the most likely word each step). Feed each predicted word back into the decoder until it predicts the end token or reaches max length — like chatting one word at a time.

## Visualizing and Enhancing Encoder-Decoder

## Task 6

Q1- Modify your decoder to include attention on encoder outputs ?

Ans - To add attention, the decoder looks at all encoder hidden states—not just the final one—so it knows where to focus for each word it generates. It calculates attention weights to highlight important parts of the input at every step. This makes translations smarter, like shining a spotlight on key words while speaking.

Q2- Visualize attention weights (e.g., with heatmaps) ?

Ans - Visualizing attention weights is like seeing *where the model is looking* while it translates each word.
Heatmaps show bright spots where attention is strongest, helping you understand which input words influenced each output word.
It's like watching the model's eyes jump from one word to another, making its translation clearer and more transparent.

## Task 7

Q1- Plot training loss and accuracy curves using matplotlib ?

Ans- To plot training loss and accuracy, use Matplotlib to draw simple line graphs showing how these values change over epochs.

Just grab the loss and accuracy history from your training, then plot them with labels and titles.

Q2- Write observations on:

a. Overfitting

b. Underfitting

c.Training stability

Ans- Overfitting: The model does great on training data but struggles on test data — like memorizing answers without understanding.

Underfitting: The model performs poorly on both training and test data — like a student who didn't study enough.

Training Stability: If loss goes down smoothly and predictions improve steadily, your training is stable — like a learner gaining confidence with practice.

## Task 8

Q1- What are the challenges in training sequence-to-sequence models?

Ans- Training sequence-to-sequence models can be tough because long sequences are hard to remember (vanishing gradients). They also need lots of data to learn good translations and can be slow to train.

Q2- What does a "bad" translation look like? Why might it happen?

Ans- A "bad" translation might sound broken, miss key words, or lose the original meaning — like "I am happy" becoming "I sad go."

It can happen if the model didn’t learn grammar well, had too little training data, or lost focus (especially without attention).

Q3- How can the model be improved further?

Ans- You can improve the model by adding attention (so it focuses better), using more training data, or switching to transformers for better performance. Tuning hyperparameters and using techniques like dropout can also help.

Assignment 2 Completed

By Anushka