DA6401 Assignment 3 (DA24M021)

Use recurrent neural networks to build a transliteration system.

Siddhant Baranwal da24m021

Created on May 15 | Last edited on May 20

Github Link: - https://github.com/Siddhant-DA24M021/da6401_assignment3.git

WandB Report Link: - wandb report

https://api.wandb.ai/links/da24m021-indian-institute-of-technology-madras/uv4jucjo

Question 1 (15 Marks)

I build a RNN based seq2seq model which contains the following layers: (i) input layer for character embeddings (ii) one encoder RNN which sequentially encodes the input character sequence (Latin) (iii) one decoder RNN which takes the last state of the encoder as input and produces one output character at a time (Devanagari).

The code is in the Github repo.

The code is flexible such that the dimension of the input character embeddings, the hidden states of the encoders and decoders, the cell (RNN, LSTM, GRU) and the number of layers in the encoder and decoder can be changed.

(a) What is the total number of computations done by your network? (assume that the input embedding size is m, encoder and decoder have 1 layer each, the hidden cell state is k for both the encoder and

decoder, the length of the input and output sequence is the same, i.e., T, the size of the vocabulary is the same for the source and target language, i.e., V)

ANS (a) :-

Assumptions: -

- Embedding Size = m
- Num of Layers = 1 (1 each for encoder and decoder)
- Hidden State Size = k
- Length of input and output sequences = T
- Vocab size of source and target languages = V
- Vanilla Model (No Attention used)

Embedding Layer:-

For one character:- O(mV)

For T characters:- O(TmV)

Encoder (Single Layer):-

- For single time step:-
  - s_i = σ(U x_i + W s_(i−1) + b)
  - U x_i → k * (m*m multiplications + (m-1) additions) = O(km^2) Computations
  - W s_(i−1) → k * (k*k multiplications + (k-1) additions) = O(k^3) Computations
  - U x_i + W s_(i−1) + b → 2*k Additions = O(k) Computations
  - Total Computations = O(km^2 + k^3 + k)
- For T time steps:-
  - Total Computations = O(Tkm^2 + Tk^3 + Tk)

Decoder (Single Layer):-

- For single time step:-
  - s_i = σ(U x_i + W s_(i−1) + b)
  - y_i = O(Z s_i + c)

    - U x_i → k * (m*m multiplications + (m-1) additions) = O(km^2) Computations
    - W s_(i−1) → k * (k*k multiplications + (k-1) additions) = O(k^3) Computations
    - U x_i + W s_(i−1) + b → 2*k Additions = O(k) Computations
    - Z s_i + c→ V * (k*k multiplications + k-1 additions) + V = O(Vk^2 + Vk + V) Computations
- Total Computations = O(km^2 + k^3 + k + Vk^2 + Vk + V)
- For T time steps:-
    - Total Computations = O(Tkm^2 + Tk^3 + Tk + TVk^2 + TVk + TV)

Total Computation of the RNN Network = O(TmV + Tkm^2 + Tk^3 + Tk + TVk^2 + TVk + TV)

Ignoring less dominant terms, total computations = O(TmV + Tkm^2 + Tk^3 + TVk^2)

NOTE:- Answers may differ slightly based on computational assumptions made. Ex: TmV could be reduced to Tm and can be ignored based on the lookup algorithm used.

(b) What is the total number of parameters in your network? (assume that the input embedding size is m, encoder and decoder have 1 layer each, the hidden cell state is k for both the encoder and decoder and the length of the input and output sequence is the same, i.e., T, the size of the vocabulary is the same for the source and target language, i.e., V)

ANS (b) :-

Assumptions: -

- Embedding Size = m
- Num of Layers = 1 (1 each for encoder and decoder)
- Hidden State Size = k
- Length of input and output sequences = T
- Vocab size of source and target languages = V

RNN :-

Encoder: -

- Embedding Matrix :- mV
- Input-to-Hidden Weights: - mk
- Hidden-to-Hidden Weights: - k^2
- Hidden Layer Bias: - k
- Hidden-to-Output Weights: - kV
- Output Layer Bias: - V

Decoder: -

- Embedding Matrix :- mV
- Input-to-Hidden Weights: - mk
- Hidden-to-Hidden Weights: - k^2
- Hidden Layer Bias: - k
- Hidden-to-Output Weights: - kV
- Output Layer Bias: - V

Total Parameters in vanilla RNN = O (mV + mk + k^2 + kV + k + V)

LSTM :-

Encoder: -

- Embedding Matrix :- mV
- Input-to-Hidden Weights: - mk
- Input-to-Gates Weights: 4km
- Hidden-to-Hidden Weights: - k^2
- Hidden-to-Gates Weights: 4k^2
- Hidden Layer Bias: - k
- Gates Biases: 4k
- Hidden-to-Output Weights: - kV
- Output Layer Bias: - V

Decoder: -

- Embedding Matrix :- mV
- Input-to-Hidden Weights: - mk
- Input-to-Gates Weights: 4km
- Hidden-to-Hidden Weights: - k^2
- Hidden-to-Gates Weights: 4k^2
- Hidden Layer Bias: - k
- Hidden-to-Output Weights: - kV
- Gates Biases: 4k
- Output Layer Bias: - V

Total Parameters in vanilla RNN = O (mV + mk + k^2 + kV + k + V)

Similarly GRU will also have parameters in the same order **O (mV + mk + k^2 + kV + k + V)**

Question 2 (10 Marks)

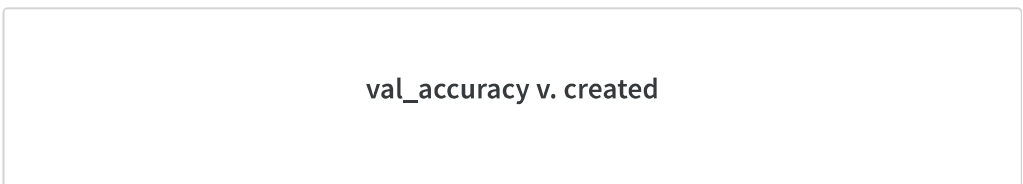
I trained my model using any one Hindi from the Dakshina dataset.

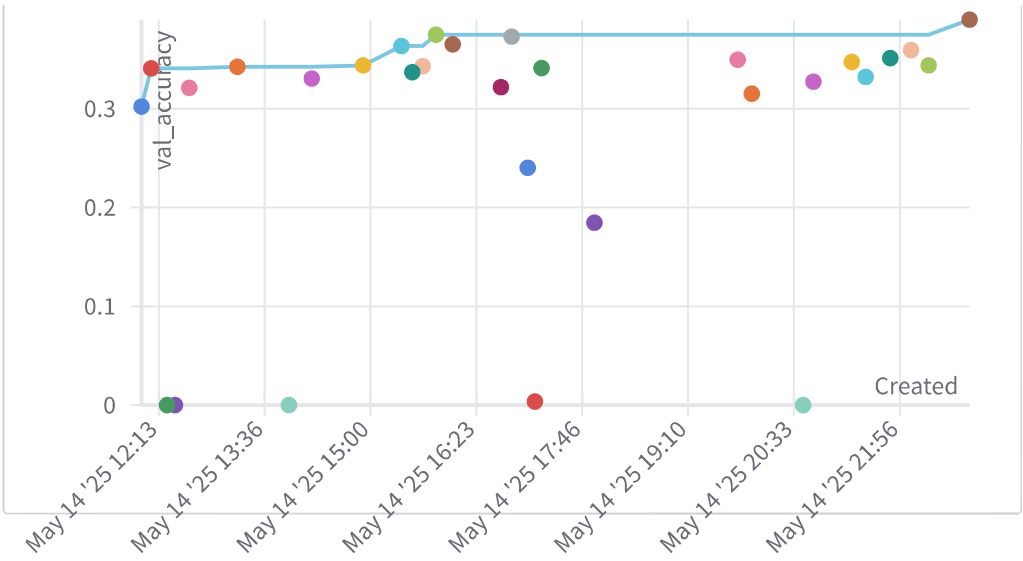
I used the following hyperparameters for my sweep

- input embedding size: 16, 32, 64, 256, 512, 1024
- number of recurrent layers: 1, 2, 3, 4
- hidden layer size: 16, 32, 64, 128, 256, 512, 1024, 2048
- cell type: RNN, GRU, LSTM
- dropout: 0%, 20%, 30%, 40%
- learning rate: 0.01, 0.001, 0.0005
- batch_size: 16, 32

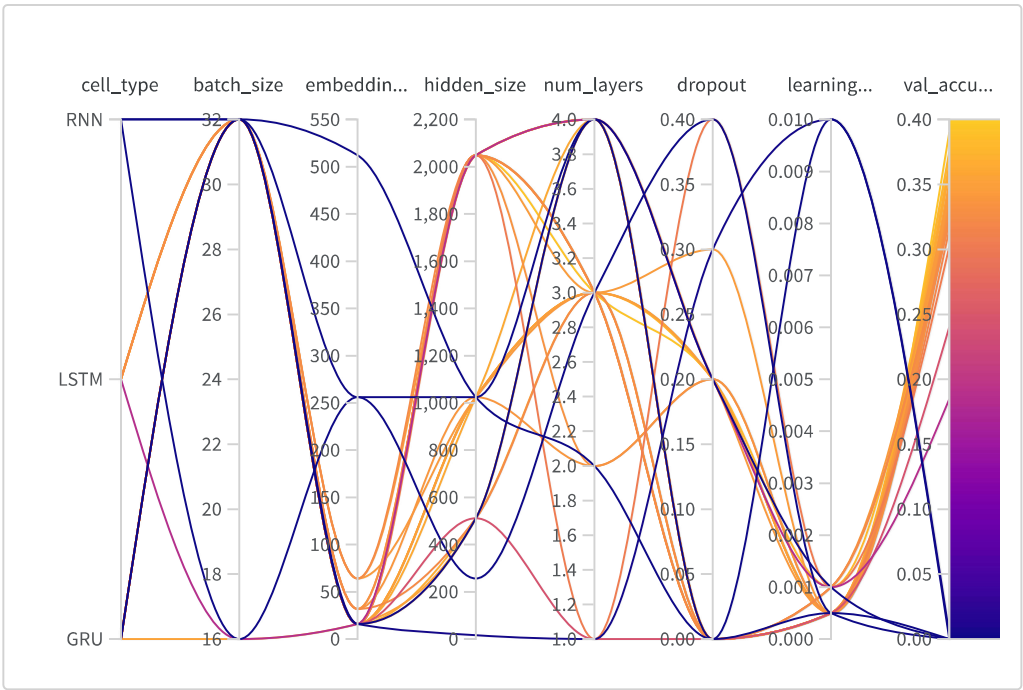
Based on sweep the following plots which are automatically generated by wandb:

- accuracy v/s created plot

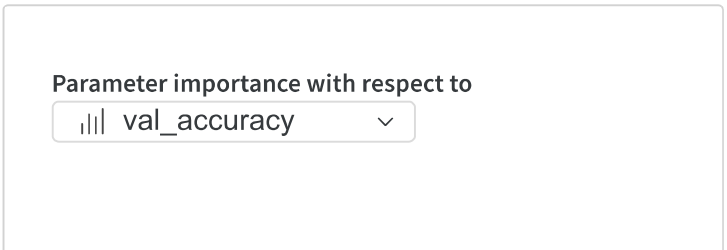


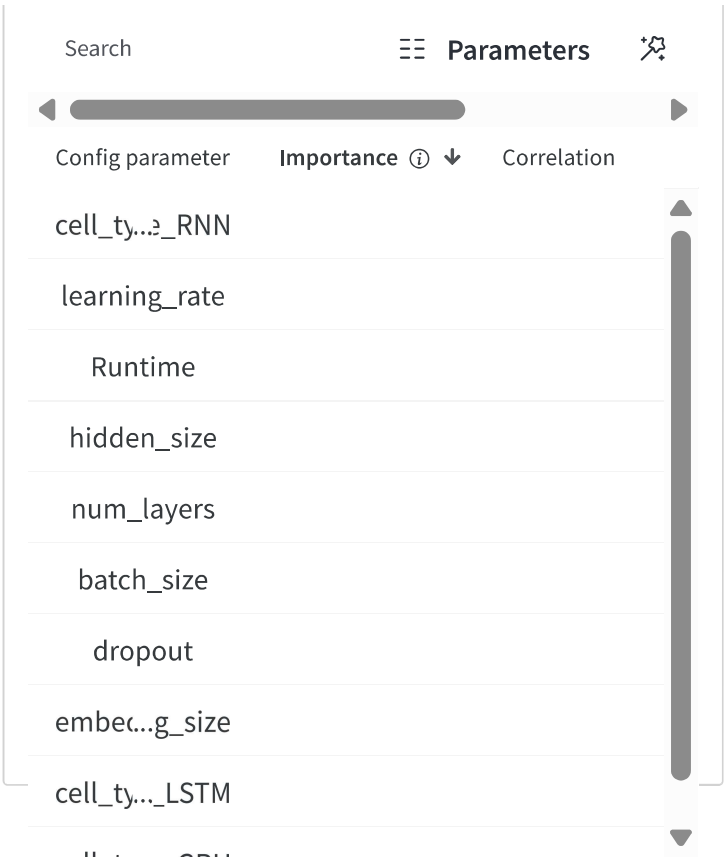


- parallel co-ordinates plot



- correlation summary table





Question 3 (15 Marks)

Based on the above plots some observations: -

- LSTM and GRU cells on average perform better than Vanilla RNN (Maybe due to increase parameters in gates).
- Smaller sizes for the hidden layer does not give good results.
- Smaller embedding layer size performs better (Maybe due to smaller embedding size got better trained).
- Dropout of 0.2 performed better than 0.3 or 0.4.
- Learning rate of 0.0005 also gave better performance than 0.01 (Stable learning and avoiding overshooting).
- 3 Recurrent layers in the encoder and decoder performs better than 1, 2 and 4 (Highlighting the fact that there needs to be a balance between too simple and too complex models).

Some other observations from the experiments: -

- Large batch sizes were giving lower train loss but the validation loss was higher. Potentially due to overfitting.
- Higher number of recurrent layers took longer to process which is expected as the sequences get passed through more number of times
- Large embedding size for characters didn't perform well as the dataset is not too big for enough training examples for each character, such that they get trained well.

Question 4 (10 Marks)

I now used my best model on the test data (I haven't used test data so far. All the above experiments have been done using train and validation data only).

(a) Use the best model from your sweep and report the accuracy on the test set (the output is correct only if it exactly matches the reference output).

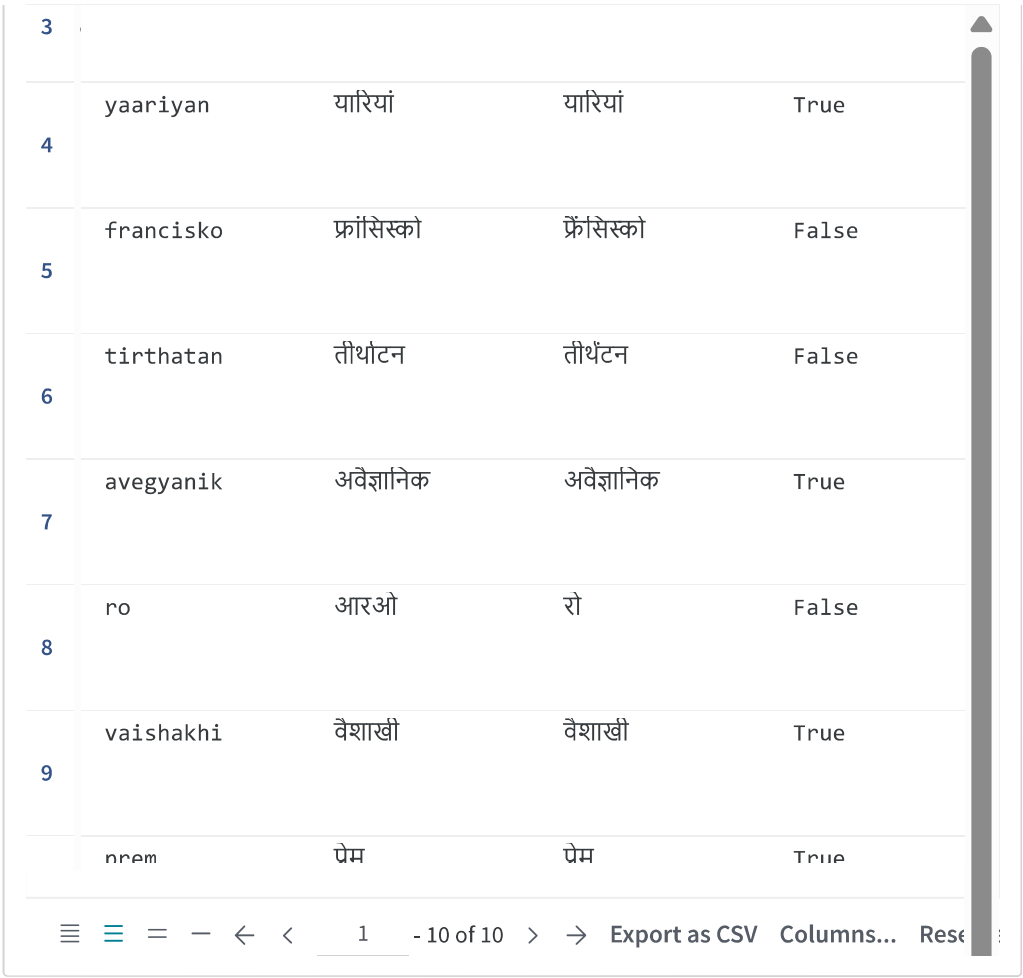
- **Train Accuracy : 77.12%**
- **Val Accuracy : 36.16%**
- **Test Accuracy : 35.78%**

(b) Sample inputs from the test data and predictions made by my best model are shown below. I also uploaded all the predictions on the test set in a folder **predictions_vanilla** on my github project.

```
runs.summary["Random Test Predictions Sample"]
```

Filter

	Latin	Correct Native	Predicted Native	Correct
1	alind	अलिंद	एलिंड	False
2	sataane	सताने	सताने	True
	sankhyayen	संख्यायें	संख्यायें	True



3				
4	yaariyan	यारियां	यारियां	True
5	francisko	फ्रांसिस्को	फ्रेंसिस्को	False
6	tirthatan	तीर्थाटन	तीर्थंटन	False
7	avegyanik	अवैज्ञानिक	अवैज्ञानिक	True
8	ro	आरओ	रो	False
9	vaishakhi	वैशाखी	वैशाखी	True
	prem	प्रेम	प्रेम	True

1 - 10 of 10 Export as CSV Columns... Rese

(c) Comment on the errors made by my model

- The model makes more errors on longer sequences
  - "antarmukh" is transliterated as "अंतरमुख" instead of "अंतर्मुख"
  - "inhaletion" is transliterated as "इंहेशेशन" instead of "इनहेलेशन"
- The model makes errors on phonetically similar sequences
  - "east" is transliterated as "ईस्ट" instead of "ईष्ट"
  - "udar" is transliterated as "उदार" instead of "उदर"
- Vowels appearing single time ("a", "o") are incorrectly translated more often than when they appear together ("aa", "ao")
  - "ubal" is transliterated as "उबल" instead of "उबाल" while "ubaal" is transliterated correctly as "उबाल"
- Similar sounding 'depending vowels' (hindi matra) are often mixed rising error. "ि"

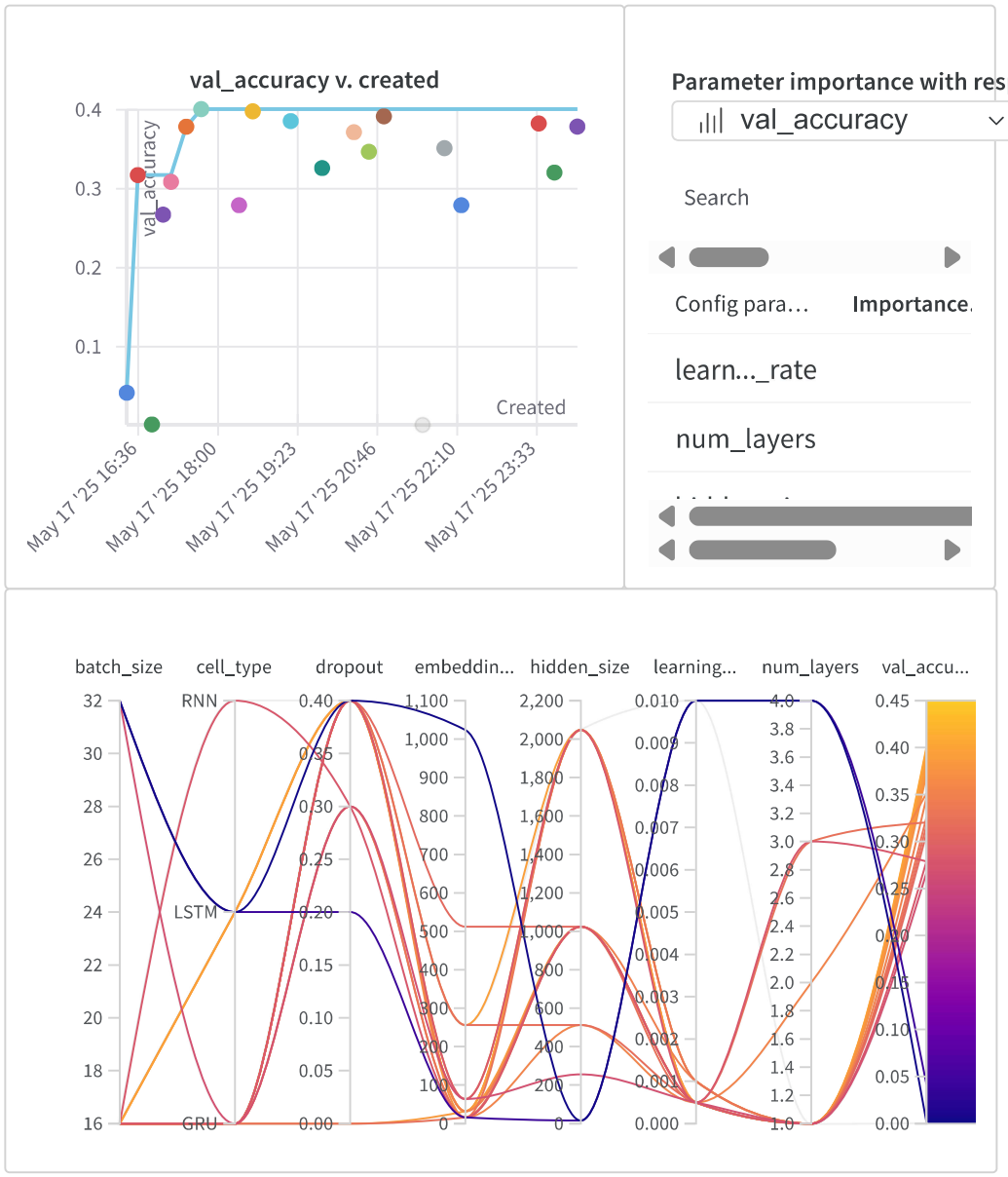
and "ी" are often intermixed. Similarly "ु" and " ू" are sometimes intermixed too.

- "engineer" is transliterated as "इंजीनियर" instead of "इंजिनियर"

Question 5 (20 Marks)

I added an attention network to my basisc sequence to sequence model and trained the model again.

(a) Plots for hyperparameters tuning:



(b) On my best attention based seq2seq model, I got the following accuracies:

Train Accuracy : 81.89%

Val Accuracy : 40.25%

Test Accuracy : 40.52%

All the predictions on the test set is uploaded in a folder **predictions_attention** on the github.

(c) Yes , the attention-based model outperforms the vanilla seq2seq model by 4% on the Test Dataset.

Some corrections made by Attention based Seq2Seq model over Vanilla Seq2Seq model are:-

1. Latin: ank

   Vanilla: एनके → Wrong

   Attention: अंक → Correct

2. Latin: andheri

   Vanilla: आंधेरी → Wrong

   Attention: अंधेरी → Correct

3. Latin: ambani

   Vanilla: अमबानी → Wrong

   Attention: अंबानी → Correct

4. Latin: hoshangabad

   Vanilla: होंशाबग → Wrong

   Attention: होशंगाबाद → Correct

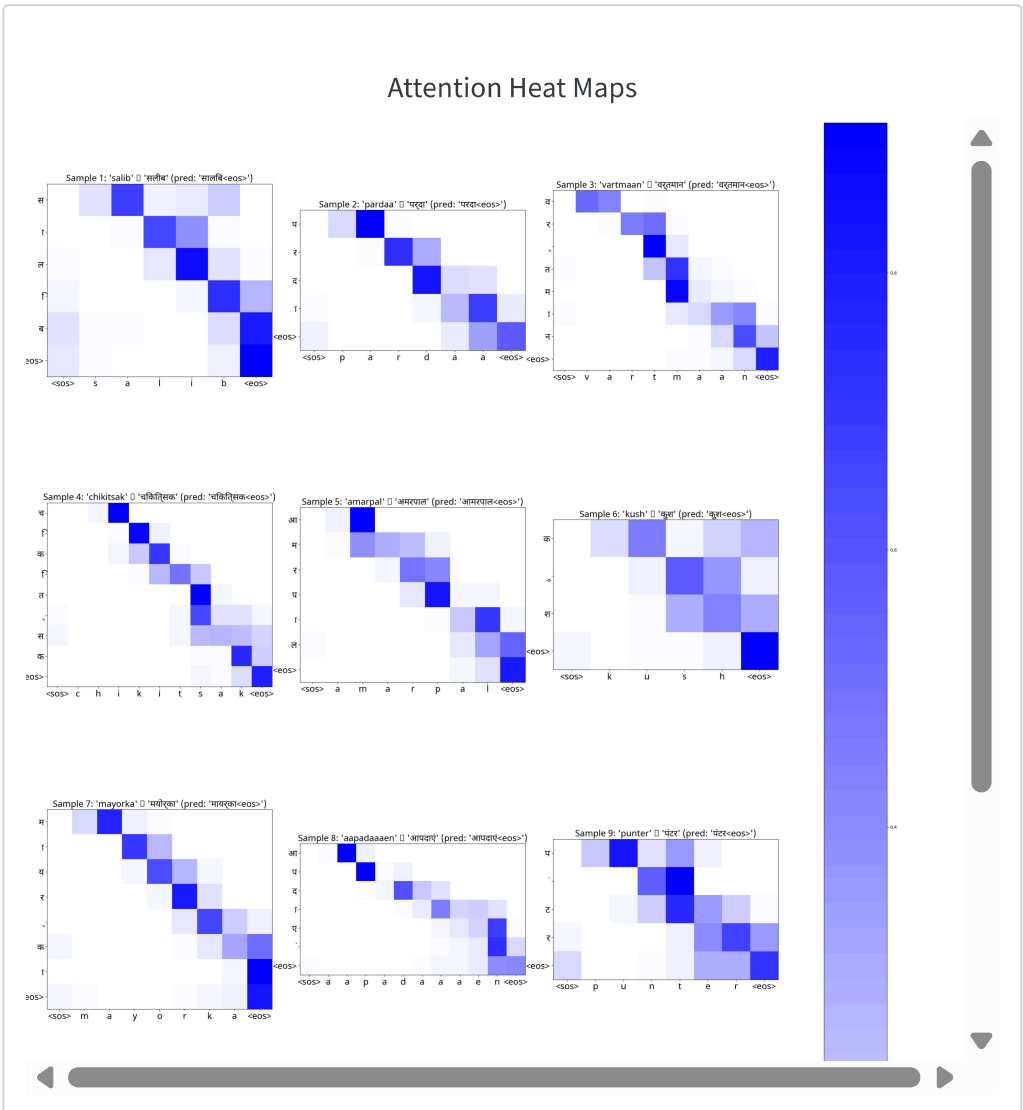
5. Latin: antrmukh

Vanilla: अंतरमुख → Wrong

Attention: अंतर्मुख → Correct

- The attention model better handles context-dependent and phonetically subtle distinctions than vanilla model.
- Attention based model shows improved alignment between input-output characters, especially in longer or more ambiguous sequences.

(d) The attention heatmaps for 10 inputs from the test data:



Question 7 (10 Marks)

Github Link: - https://github.com/Siddhant-DA24M021/da6401_assignment3.git

Self Declaration

I, Siddhant Baranwal (Roll no: DA24M021), swear on my honour that I have written the code and the report by myself and have not copied it from the internet or other students.