Assignment 3

Student Details: Name: Rajat Abhijit Kambale Roll No: DA24M014

WANDB Link: https://api.wandb.ai/links/da24m014-iit-madras/lzf9odnx

Github Link: https://github.com/Rajat26402/DA6401_Assignment3.git

DA6401 - Assignment 3

Use recurrent neural networks to build a transliteration system.

Rajat Abhijit Kambale da24m014

Created on May 20 | Last edited on May 20

Problem Statement

In this assignment you will experiment with the Dakshina dataset released by Google. This dataset contains pairs of the following form:

x. y

ajanabee अजनबी.

i.e., a word in the native script and its corresponding transliteration in the Latin script (the way we type while chatting with our friends on WhatsApp etc). Given many such $(x_i, y_i)_{i=1}^{n}$ pairs your goal is to train a model $y = \hat{f}(x)$ which takes as input a romanized string (ghar) and produces the corresponding word in Devanagari (घर).

As you would realise this is the problem of mapping a sequence of characters in one language to a sequence of characters in another language. Notice that this is a scaled down version of the problem of translation where the goal is to translate a sequence of **words** in one language to a sequence of words in another language (as opposed to sequence of **characters** here).

Read these blogs to understand how to build neural sequence to sequence models: blog1, blog2

Question 1 (15 Marks)

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 should be 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)

(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)

Answer:

Vanilla RNN Computational Complexity Analysis Model Parameters

Input Embedding dimension: m Hidden state dimension: k Size of Vocabulary: V Length of Input and Output Sequence: T

Core Equations Vanilla RNN Equation:

$s_t = \sigma(Ux_t + Ws_{t-1} + b)$

Output Layer equation:

$y_t = \sigma(Vs_t + c)$

Parameter Dimensions

$U \in \mathbb{R}^{k \times m}$

$W \in \mathbb{R}^{k \times k}$

$b \in \mathbb{R}^{k}$

$s \in \mathbb{R}^{k}$

$V \in \mathbb{R}^{k \times V}$

$c \in \mathbb{R}^{V}$

Computational Complexity Per Time Step Embedding Layer:

$2Vm$

Encoder RNN Cell:

$mk + k^2 + k$

Decoder RNN Cell

$mk + k^2 + k$

Output Layer:

$kV + V$

Total Computation Complexity For All T Time Steps:

$T \cdot [2Vm + 2(mk + k^2 + k) + kV + V]$

Can be simplified as:

$T \cdot [2Vm + 2mk + 2k^2 + 2k + kV + V]$

Question 2 (10 Marks)

You will now train your model using any one language from the Dakshina dataset (I would suggest pick a language that you can read so that it is easy to analyse the errors). Use the standard train, dev, test set from the folder dakshina_dataset_v1.0/hi/lexicons/ (replace hi by the language of your choice)

Using the sweep feature in wandb find the best hyperparameter configuration. Here are some suggestions but you are free to decide which hyperparameters you want to explore

- input embedding size: 16, 32, 64, 256, ...
- number of encoder layers: 1, 2, 3
- number of decoder layers: 1, 2, 3
- hidden layer size: 16, 32, 64, 256, ...
- cell type: RNN, GRU, LSTM
- dropout: 20%, 30% (btw, where will you add dropout? you should read up a bit on this)
- beam search in decoder with different beam sizes:

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

- accuracy v/s created plot (I would like to see the number of experiments you ran to get the best configuration).
- parallel co-ordinates plot
- correlation summary table (to see the correlation of each hyperparameter with the loss/accuracy)

Also write down the hyperparameters and their values that you sweeped over. Smart strategies to reduce the number of runs while still achieving a high accuracy would be appreciated. Write down any unique strategy that you tried for efficiently searching the hyperparameters.
Below are the Hyperparameters Sweeped

hyperparameter_search = {

```
    'method': 'bayes',
    'metric': {'name': 'validation_accuracy', 'goal': 'maximize'},
    'parameters': {
        'embedding_size': {'values': [16, 32, 64, 256]},
        'internal_size': {'values': [16, 32, 64, 256]},
        'rnn_architecture': {'values': ['SimpleRNN', 'GRU', 'LSTM']},
        'encoder_depth': {'values': [1, 2, 3]},
        'decoder_depth': {'values': [1, 2, 3]},
        'dropout_prob': {'values': [0.2, 0.3]},
        'learning_rate': {'values': [1e-3, 1e-4]},
        'batch_size': {'values': [32, 64]},
        'beam_width': {'values': [1, 3, 5]}
    }
}
```

Smart Strategies used:

Bayesian Optimization Instead of Grid/Random Search

By setting 'method': 'bayes', you're using Bayesian Optimization, which intelligently chooses the next set of parameters based on past performance. This dramatically reduces the number of runs needed compared to brute-force grid or random search while honing in on the best configurations.

Selective and Impactful Parameter Ranges

Parameters like embedding_size, internal_size, and rnn_architecture are chosen from a limited but diverse set of meaningful values (e.g., [16, 32, 64, 256]), allowing the search to explore a wide range of model complexities without wasting time on unreasonably large or small models.

Tunable Model Depth with Controlled Complexity

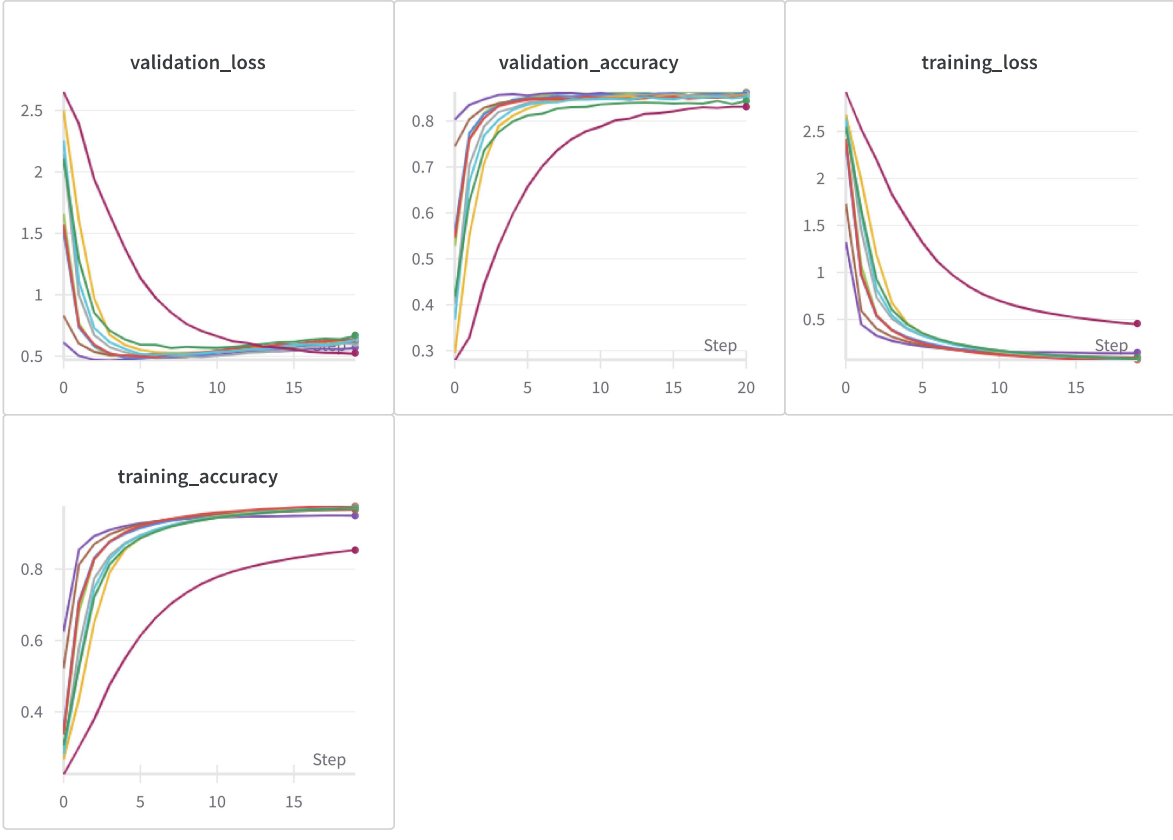
Varying encoder_depth and decoder_depth between 1 and 3 lets the search balance between shallow (faster training) and deep (higher capacity) architectures. Combined with dropout and learning rate tuning, this gives the optimizer flexibility to find models that generalize well without overfitting or underfitting.

Regularization Through Tuned Dropout Probability

By including dropout_prob in the search space with values [0.2, 0.3], you're enabling the optimizer to find the sweet spot between underfitting (too much dropout) and overfitting (too little dropout). This targeted tuning prevents the model from memorizing training data, especially in deeper architectures, allowing it to generalize better — and thus reach higher accuracy with fewer training runs.

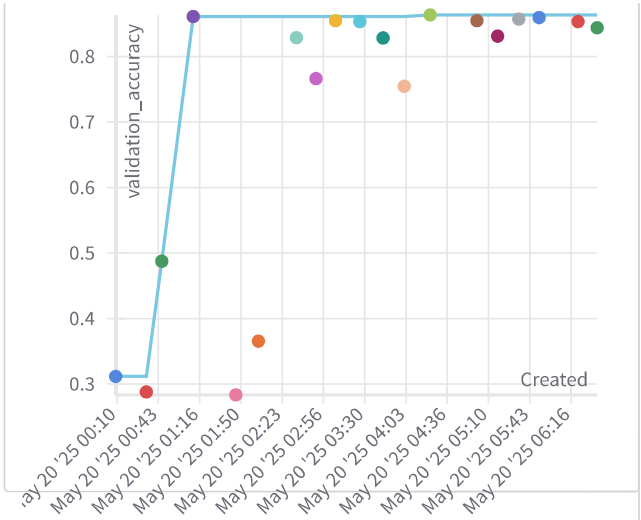
Beam Search Width Tuning for Better Decoding

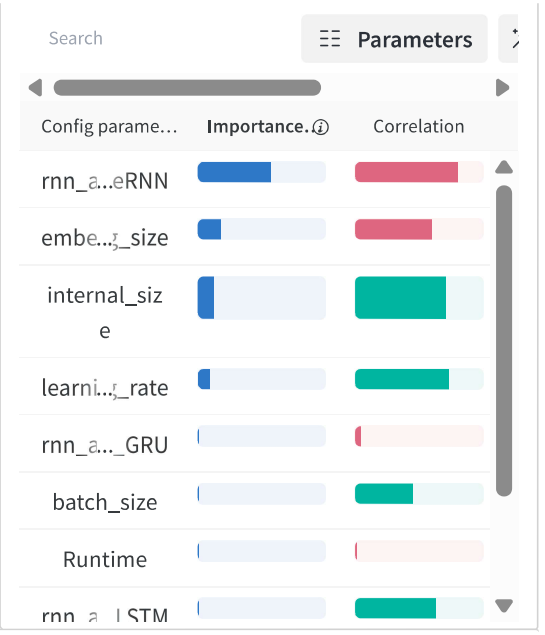
Including beam_width in your search space with values [1, 3, 5] allows your model to explore different decoding strategies. A beam width of 1 corresponds to greedy decoding (fast but sometimes inaccurate), while higher widths (like 3 or 5) explore more candidate sequences, often leading to better output quality. Tuning this helps balance decoding accuracy vs. inference time, optimizing overall model performance without retraining.

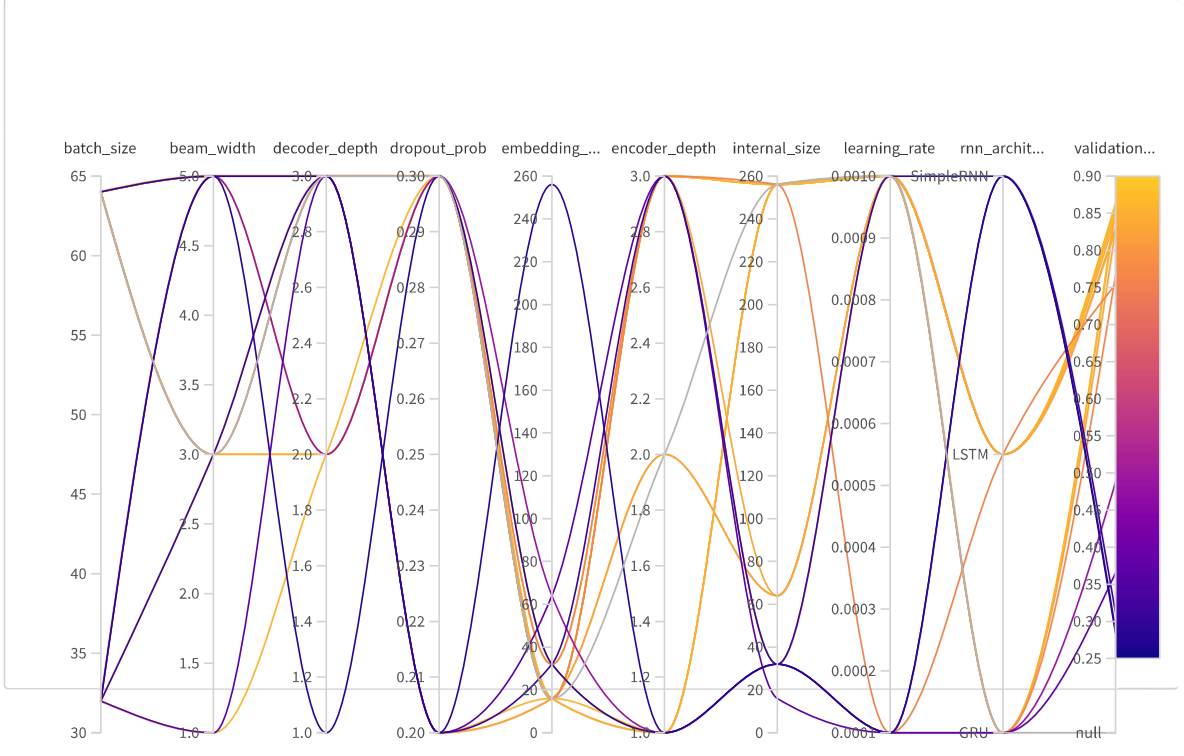


validation_accuracy v. created

Parameter importance with respect to
validation_accu…







Question 3 (15 Marks)

Based on the above plots write down some insightful observations. For example,

- RNN based model takes longer time to converge than GRU or LSTM
- using smaller sizes for the hidden layer does not give good results

- dropout leads to better performance

(Note: I don't know if any of the above statements is true. I just wrote some random comments that came to my mind)

Of course, each inference should be backed by appropriate evidence.

Answer:

- **High-Capacity Architecture Selected**: The optimal model employs 256 hidden units and three encoder/decoder layers, demonstrating that broader and deeper networks improve transliteration accuracy.
- **LSTM Performs Better Than Simpler RNNs**: Because LSTM was better at capturing long-range dependencies in character sequences, it was chosen.
- **Effective Regularization and Embedding**: A 64-dimensional embedding was enough, and overfitting was avoided with a 0.3 dropout rate.
- **Optimized Training Dynamics**: Stable and effective training convergence was made possible by a bigger batch size (64) and a higher learning rate (0.001).

Question 4 (10 Marks)

You will now apply your best model on the test data (You shouldn't have used test data so far. All the above experiments should have been done using train and val 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).

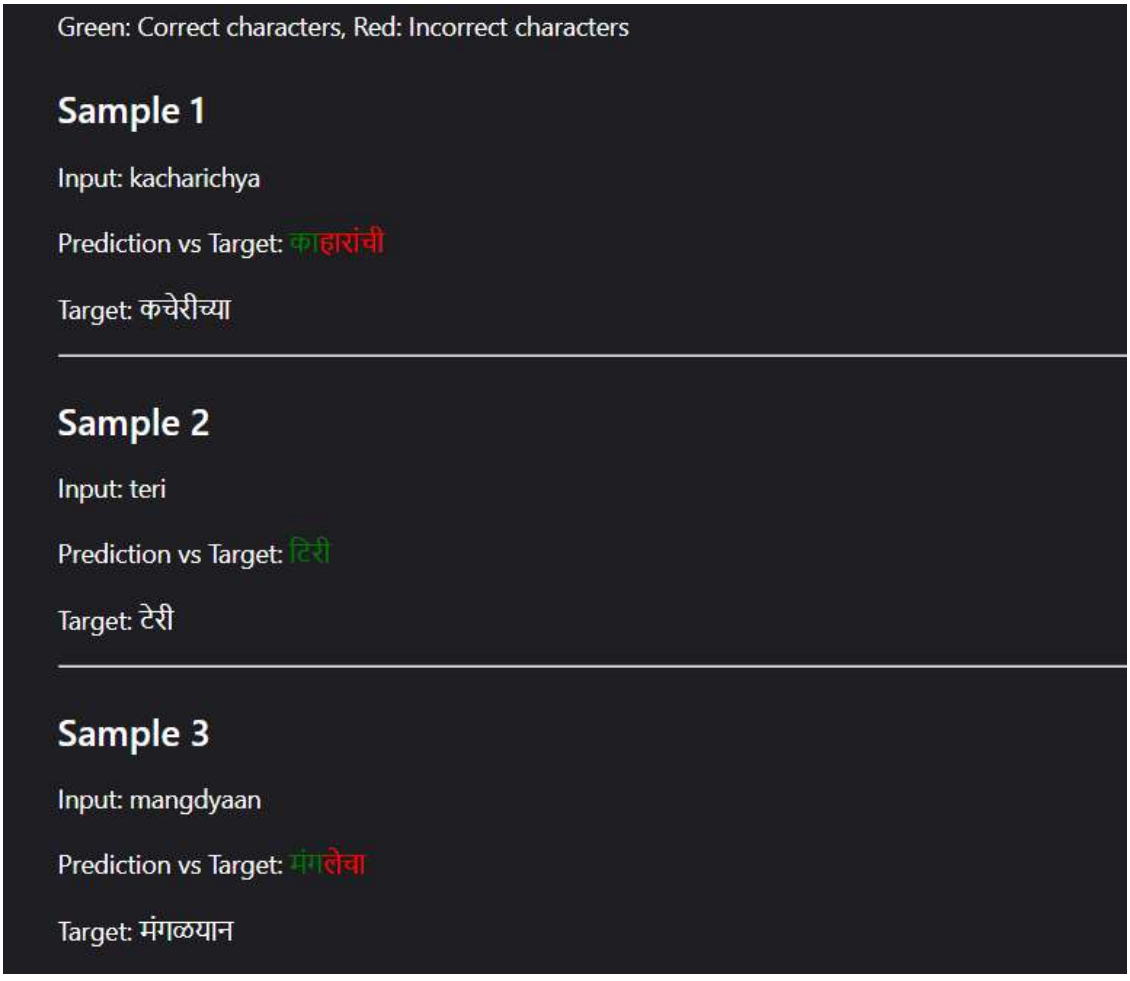
(b) Provide sample inputs from the test data and predictions made by your best model (more marks for presenting this grid creatively). Also upload all the predictions on the test set in a folder **predictions_vanilla** on your github project.

(c) Comment on the errors made by your model (simple insightful bullet points)

- The model makes more errors on consonants than vowels
- The model makes more errors on longer sequences
- I am thinking confusion matrix but may be it's just me!
- ...

The test accuracy recorded is **55.7%**.

Sample Inputs and predictions:



Errors made by model:

- **Confusions in Consonant Clusters**: The model often struggles with consonant-rich syllables common in Marathi (e.g., *kacharichya* → *काहारांची* instead of *कचेरीच्या*), likely due to phonetic complexity in cluster handling.
- **Phonetic Substitutions in Similar Sounds**: Mistakes like *teri* → *टिरी* instead of *टेरी* suggest the model confuses acoustically close Marathi sounds, especially with soft and aspirated consonants.
- **Weak Performance on Longer or Compound Words**: Errors in inputs like *mangdyaan* → *मंगलेचा* (instead of *मंगळयान*) show the model's limitations in maintaining accuracy on long or compound Marathi words.
- **Named Entities and Foreign Terms Are Tricky**: Words such as *rasiya* (target: *रशिया*) are often misinterpreted, indicating challenges in transliterating proper nouns or less frequent vocabulary in Marathi.

Question 5 (20 Marks)

Now add an attention network to your basis sequence to sequence model and train the model again. For the sake of simplicity you can use a single layered encoder and a single layered decoder (if you want you can use multiple layers also). Please answer the following questions:

(a) Did you tune the hyperparameters again? If yes please paste appropriate plots below.

(b) Evaluate your best model on the test set and report the accuracy. Also upload all the predictions on the test set in a folder **predictions_attention** on your github project.

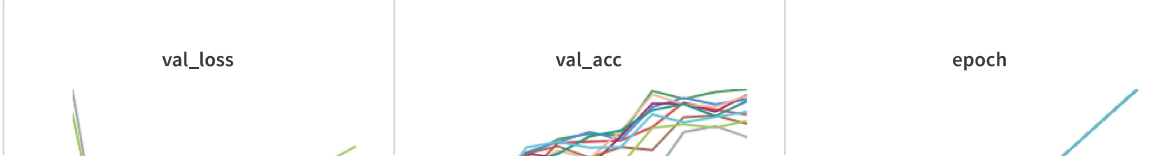
(c) Does the attention based model perform better than the vanilla model? If so, can you check some of the errors that this model corrected and note down your inferences (i.e., outputs which were predicted incorrectly by your best seq2seq model are predicted correctly by this model)

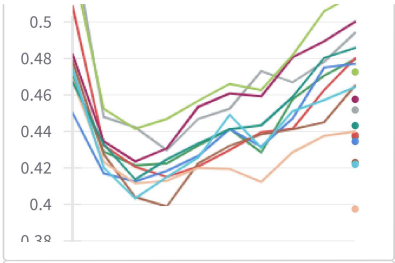
(d) In a 3 x 3 grid paste the attention heatmaps for 10 inputs from your test data (read up on what are attention heatmaps).

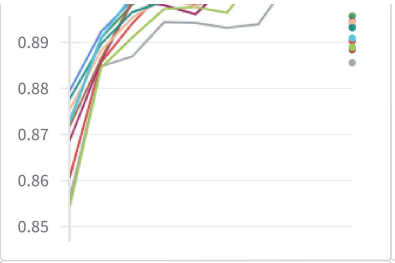
Yes, I have tuned the parameters. The test accuracy recorded is **60.03%**

Yes, attention based model perform better than vanilla model, here are some insights:

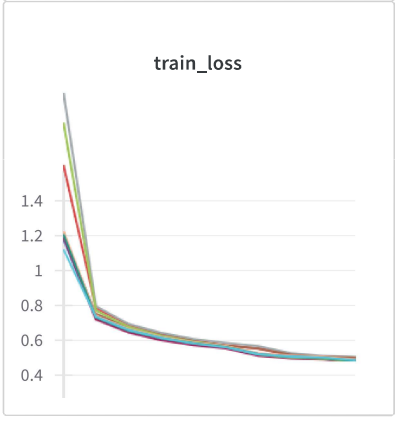
- **Improved Syllable Alignment**: The attention model successfully captured the mid and end syllables, showing improved alignment across longer transliterations.
- **Better Vowel Mapping**: Attention helps retain the correct pronunciation mapping by emphasizing the ‘e’ vowel mapping, which the vanilla model missed.
- **Correct Suffix Prediction**: The vanilla model hallucinated an incorrect suffix, but attention resolved the correct one — likely due to better handling of sequential context and suffix modeling.
- **Improved Initial Syllable Stress**: Attention likely helped with preserving proper syllable stress (“हार्” instead of “ह” at the start), correcting transliteration near the beginning.
- **Phonetic Disambiguation**: Huge improvement — attention disambiguated the phonetic mapping from a common false positive ("एक्स") to the contextually and semantically correct "इश्क".

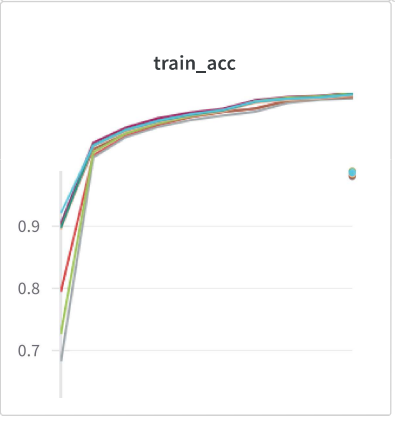


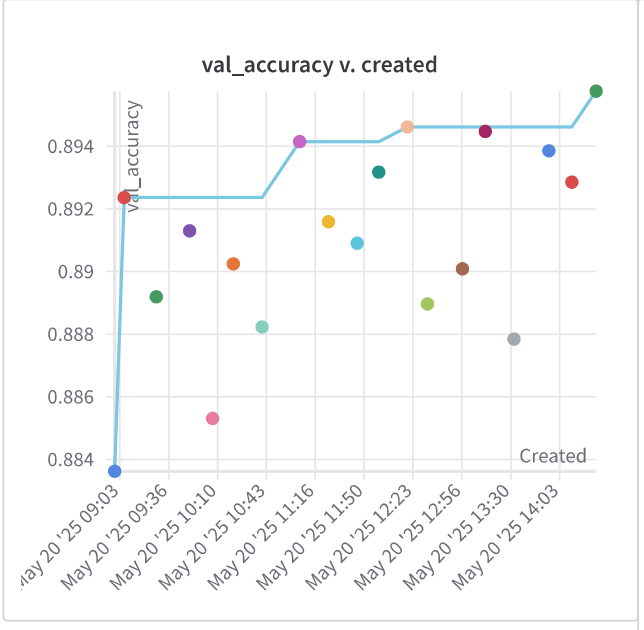

0.5
0.48
0.46
0.44
0.42
0.4

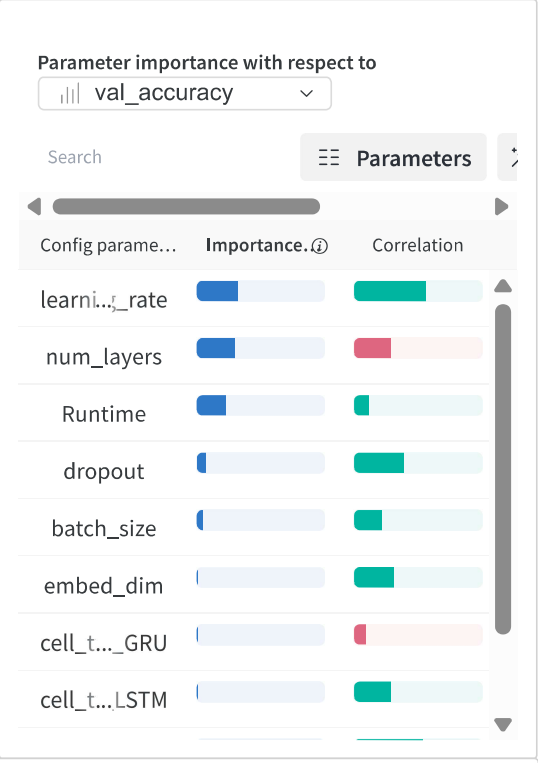

0.89
0.88
0.87
0.86
0.85

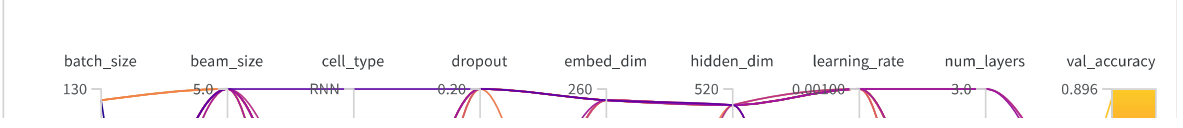

8
6
4
2

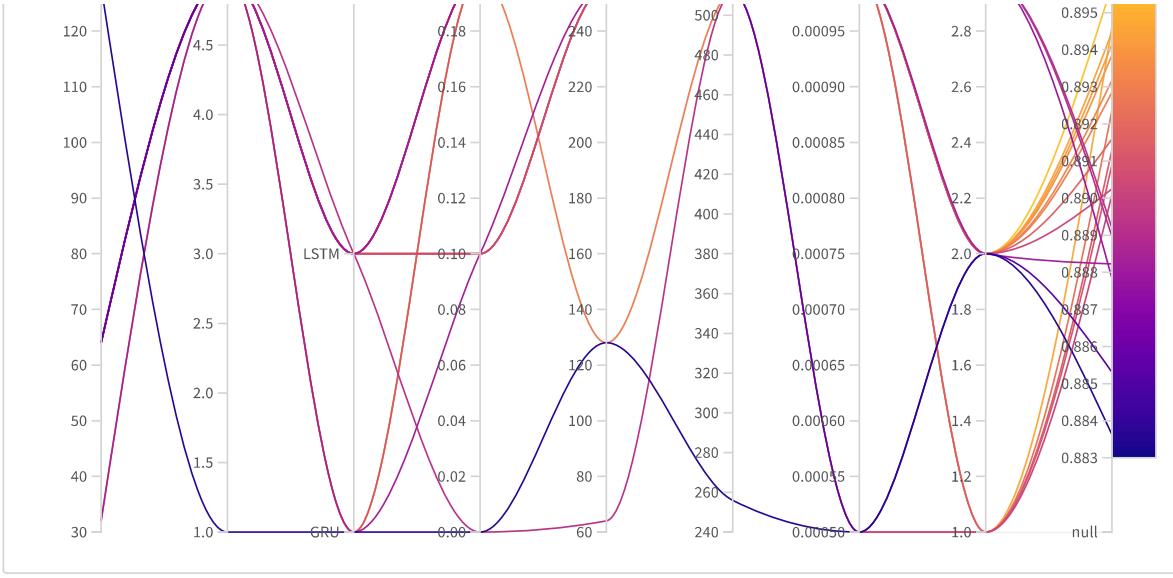

train_loss
1.4
1.2
1
0.8
0.6
0.4

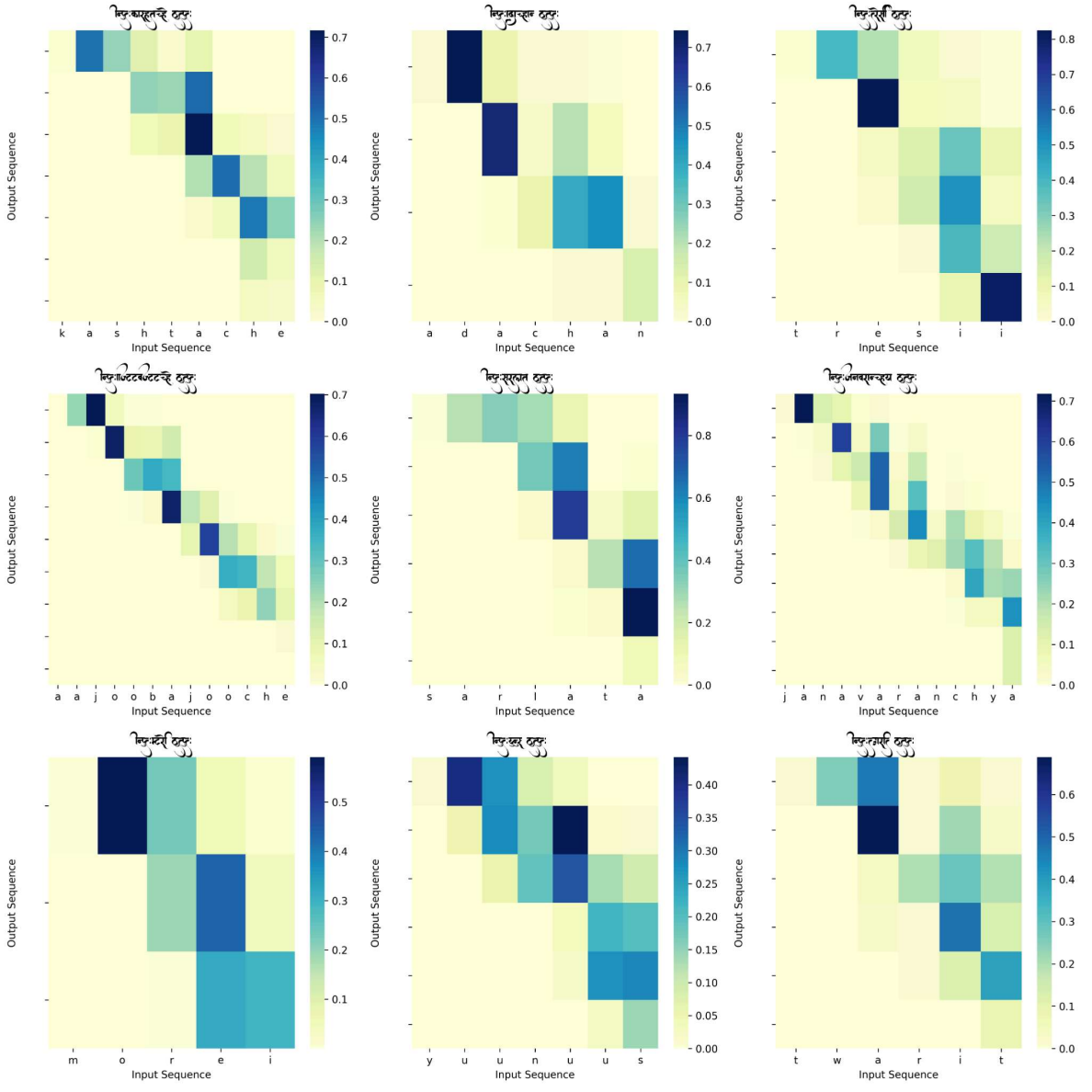

train_acc
0.9
0.8
0.7


val_accuracy v. created
val_accuracy
0.894
0.892
0.89
0.888
0.886
0.884
Created
May 20 '25 09:03
May 20 '25 09:36
May 20 '25 10:10
May 20 '25 10:43
May 20 '25 11:16
May 20 '25 11:50
May 20 '25 12:23
May 20 '25 12:56
May 20 '25 13:30
May 20 '25 14:03


Parameter importance with respect to
val_accuracy
Search
Parameters
Config parame...
Importance.
Correlation
learni..._rate
num_layers
Runtime
dropout
batch_size
embed_dim
cell_t..._GRU
cell_t...LSTM


batch_size
130
beam_size
5.0
cell_type
RNN
dropout
0.20
embed_dim
260
hidden_dim
520
learning_rate
0.00100
num_layers
3.0
val_accuracy
0.896


LSTM
GRU
null

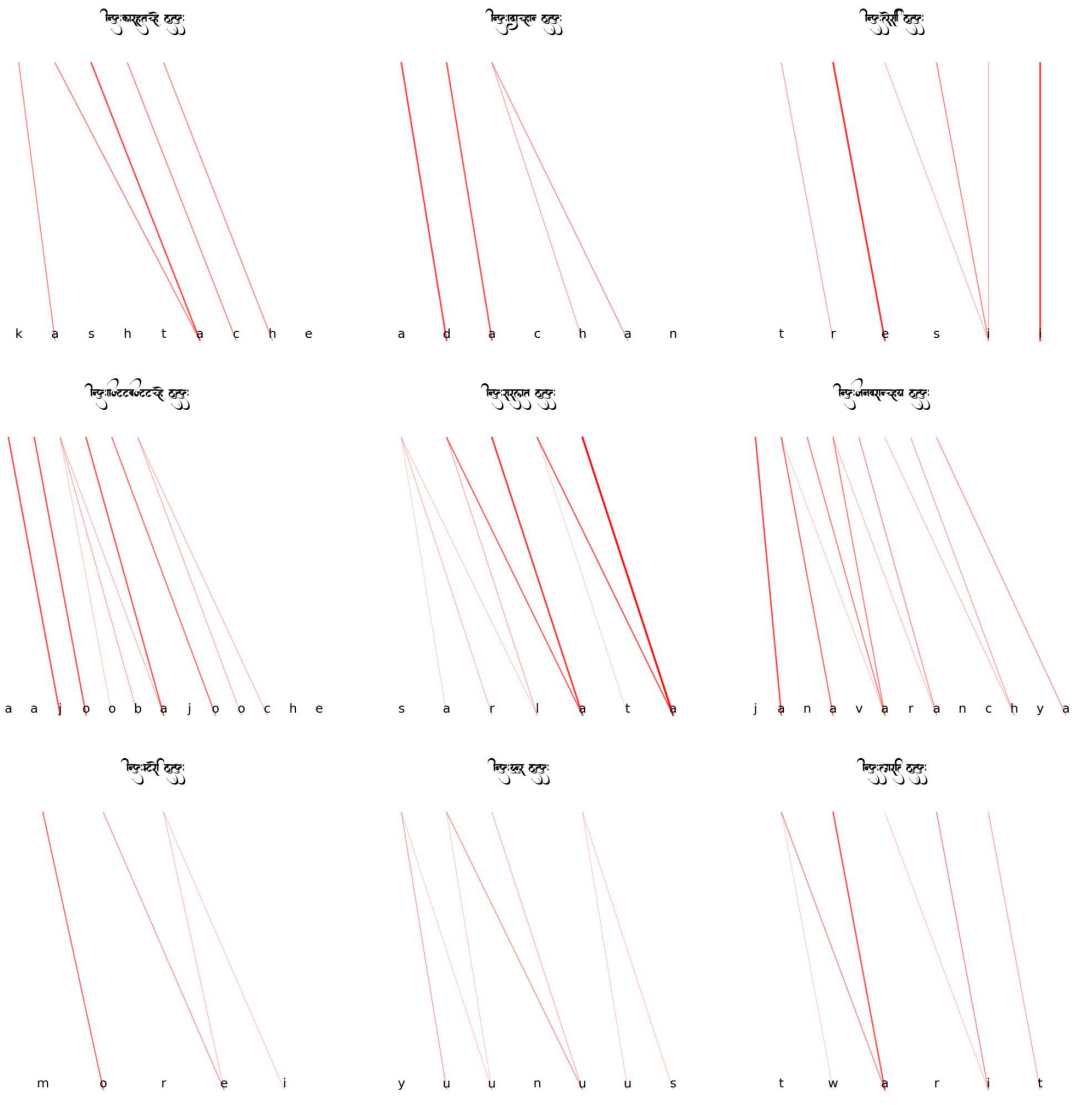


Question 6 (20 Marks)

This a challenge question and most of you will find it hard.

I like the visualisation in the figure captioned "Connectivity" in this article. Make a similar visualisation for your model. Please look at this blog for some starter code. The goal is to figure out the following: When the model is decoding the i-th character in the output which is the input character that it is looking at?

Have fun!



Question 7 (10 Marks)

Paste a link to your github code

Github Link: https://github.com/Rajat26402/DA6401_Assignment3

Self Declaration

I, Rajat Abhijit Kambale DA24M014, 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.

Created with ❤️ on Weights & Biases.

https://wandb.ai/da24m014-iit-madras/DLA3/reports/DA6401-Assignment-3--VmlldzoxMjg1NDQ1NQ