Assignment 3

Student Details: Name:Kaniganti Priyanka Saraswathi Roll No: DA24M013

WANDB Link: https://api.wandb.ai/links/da24m013-iit-madras-alumni-association/tn2ksqt4

Github Link: https://github.com/PriyankaDA24M013/DA6401_Assignment-3.git

Submission Instructions : kindly format your doc like this page (with your details and links) and then submit it.

Important Instructions:

- Students **must follow the updated submission format strictly**. Non-compliance will result in penalties.
- **Submitting non-modular, unstructured, or unreadable code will cause penalties**.
- Please upload all your `.py` files and the original `.ipynb` notebooks to GitHub, along with a detailed README file that explains the assignment structure, functionality, and instructions for running the code. Any deviation from it will cause penalties.
- You will be heavily penalized if we can't access your GitHub repo and W&B links.
- Any plagiarism will be reported and heavily penalized.
- **Note that the deadlines are stringent for this assignment, as grades need to be finalized before the institute-mandated schedule.**

Share Comment Star

DA6401 Assignment 3

Use recurrent neural networks to build a transliteration system.

Priyanka Saraswathi Kaniganti

Created on May 18 | Last edited on May 20

Instructions

- The goal of this assignment is fourfold: (i) learn how to model sequence to sequence learning problems using Recurrent Neural Networks (ii) compare different cells such as vanilla RNN, LSTM and GRU (iii) understand how attention networks overcome the limitations of vanilla seq2seq models (iv) visualise the interactions between different components in a RNN based model.
- Collaborations and discussions with other groups are strictly prohibited.
- You must use Python (numpy and pandas) for your implementation.
- You can use any and all packages from keras, pytorch, tensorflow
- You can run the code in a jupyter notebook on colab by enabling GPUs.
- You have to generate the report in the same format as shown below using wandb.ai. You can start by cloning this report using the clone option above. Most of the plots that we have asked for below can be (automatically) generated using the apis provided by wandb.ai. You will upload a link to this report on gradescope.
- You also need to provide a link to your github code as shown below. Follow good software engineering practices and set

up a github repo for the project on Day 1. Please do not write all code on your local machine and push everything to github on the last day. The commits in github should reflect how the code has evolved during the course of the assignment.

- You have to check moodle regularly for updates regarding the assignment.

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)

Component	**Computation per Time Step**	**Total Time Steps**	**Total Computations**
Embedding Layer	None (just a lookup)	T	0
Encoder RNN	$\mathrm{m} \cdot \mathrm{k} + \mathrm{k} \cdot \mathrm{k}$	T	$\mathrm{T} \cdot (\mathrm{m} \cdot \mathrm{k} + \mathrm{k}^2)$
Decoder RNN	$\mathrm{k} \cdot \mathrm{k} + \mathrm{k} \cdot \mathrm{k}$	T	$\mathrm{T} \cdot (\mathrm{k}^2 + \mathrm{k}^2) = \mathrm{T} \cdot 2\mathrm{k}^2$
Output Layer (Softmax)	$\mathrm{k} \cdot \mathrm{V}$	T	$\mathrm{T} \cdot \mathrm{k} \cdot \mathrm{V}$
Total			$\mathrm{T} \cdot (\mathrm{m} \cdot \mathrm{k} + 3\mathrm{k}^2 + \mathrm{k} \cdot \mathrm{V})$

(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)

Component	Shape/Details	Number of Parameters
Embedding Layer	$V \times m$	$V \cdot m$
Encoder RNN Weights	Input weights: $m \times k$, Hidden weights: $k \times k$, Bias: k	$m \cdot k + k \cdot k + k$
Decoder RNN Weights	Input weights: $k \times k$, Hidden weights: $k \times k$, Bias: k	$k \cdot k + k \cdot k + k = 2k^2 + k$
Output Layer (Dense + Softmax)	Weights: $k \times V$, Bias: V	$k \cdot V + V$
Total		$V \cdot m + m \cdot k + 3k^2 + 2k + k \cdot V + V$ or simplified: $V(m + k + 1) + m \cdot k + 3k^2 + 2k$

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.

- **Accuracy vs. Created**
  - Around **30 experiments** were conducted.
  - Best validation accuracy reached ≈ **0.913**.
  - Most runs performed above **0.91**, indicating consistent optimization.
- **Parallel Coordinates Plot**
  - High-accuracy models (yellow lines) often share:
    - `cell_type`: LSTM or GRU
    - `hid_dim`: ≥ 140
    - `dropout`: ~0.25–0.35
    - `n_layers`: 2
    - `beam_size`: 2 or 3
- **Correlation Table**

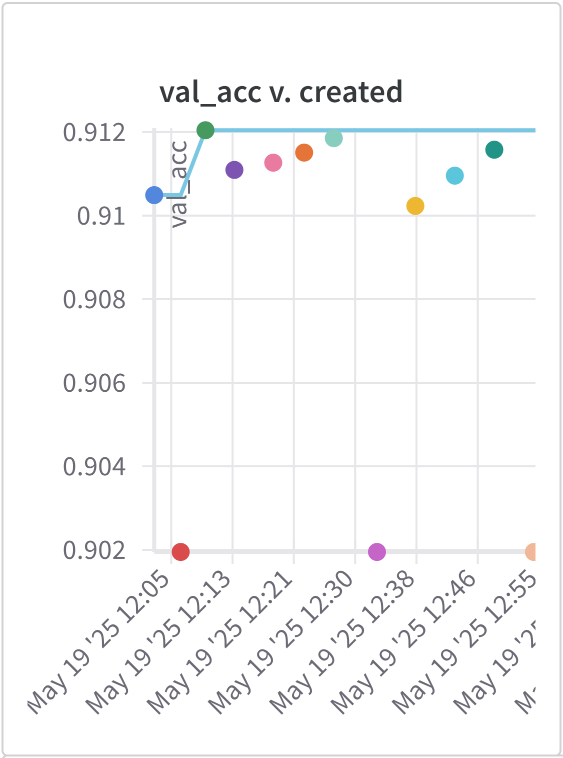
  - `cell_type` shows the **strongest correlation** with accuracy.
  - `dropout` and `hid_dim` also positively correlate.
  - `beam_size` has **minor** correlation.
  - `Runtime` correlates weakly with validation accuracy.

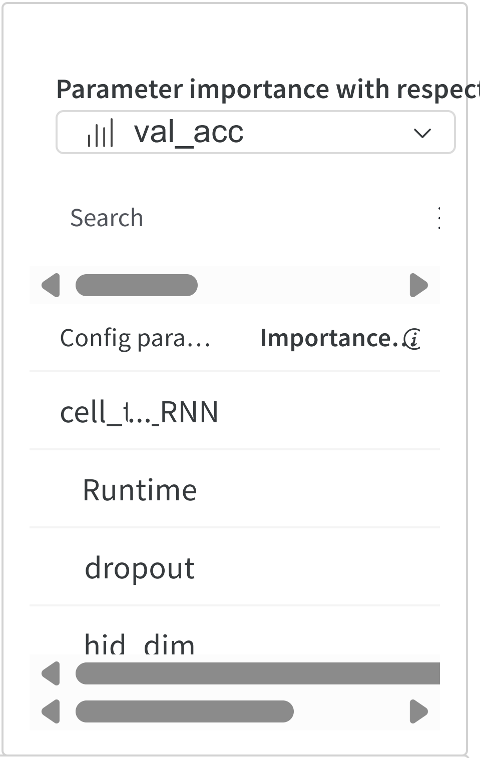
Hyperparameters and Values Swept

Hyperparameter	Values/Range Explored
`cell_type`	RNN, LSTM, GRU
`dropout`	0.2 – 0.4
`emb_dim`	60, 90, 120, 130
`hid_dim`	60, 140, 200, 260
`n_layers`	1, 2
`beam_size`	1, 2, 3

Smart Search Strategy

- **Bayesian Optimization** was used via WandB sweeps:
  - Efficiently selects promising configurations based on prior results.
  - Reduces the total number of runs needed compared to grid/random search.
- **Strategy Highlights**:
  - Prioritized exploration of LSTM/GRU after initial RNN underperformance.
  - Limited embedding and hidden dimensions to practical ranges to avoid overfitting and slow convergence.
  - Focused on key influencers (`cell_type`, `dropout`, `hid_dim`) based on early correlation feedback.

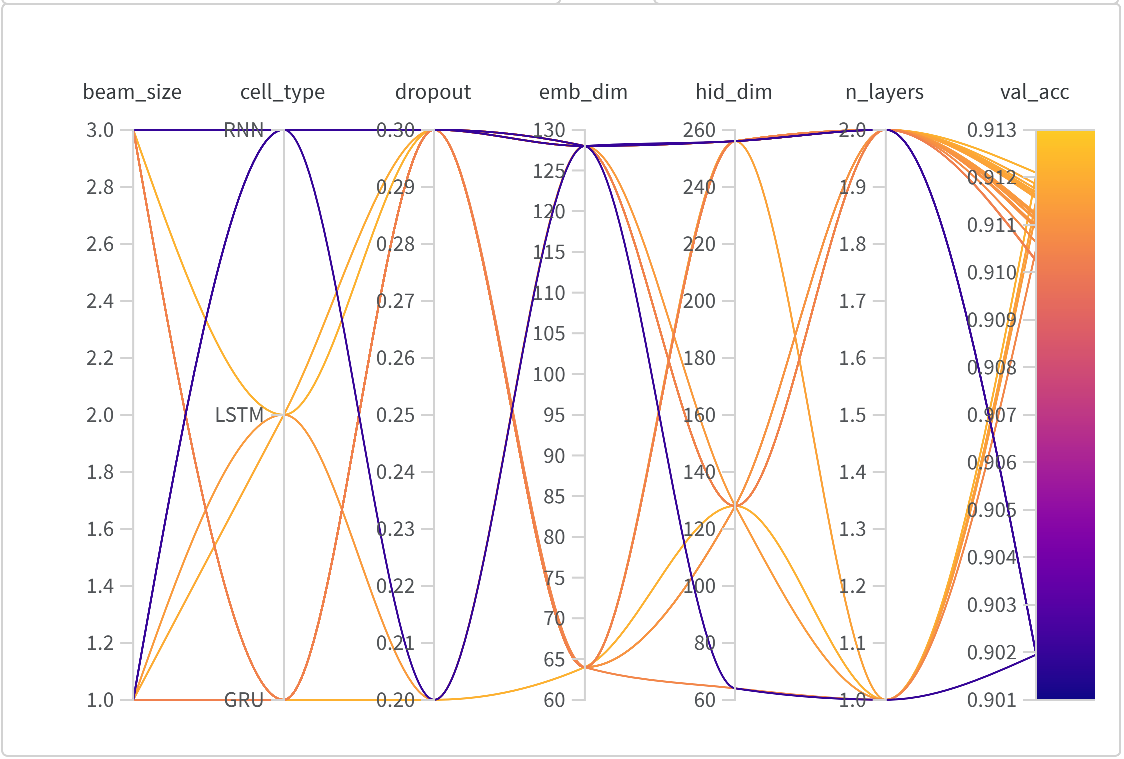



Parameter importance with respect to

val_acc

Search

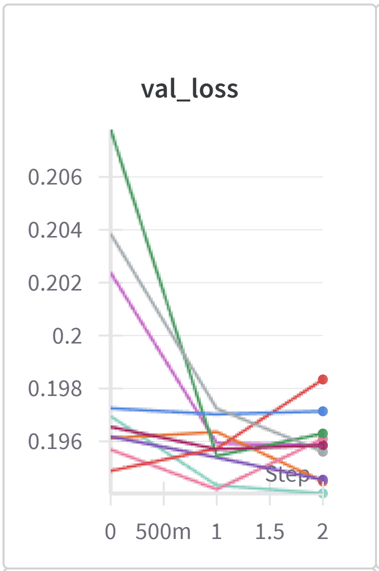
Config para...	Importance.
cell_t..._RNN	
Runtime	
dropout	
hid_dim	

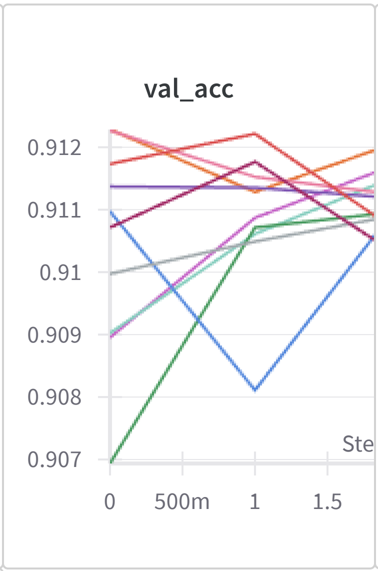


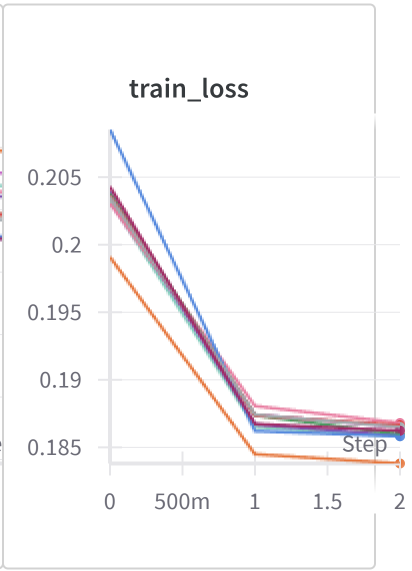
Question 3 (15 Marks)

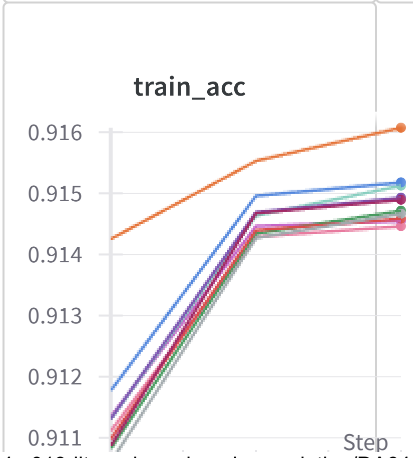
Based on the above plots write down some insightful observations. For example,

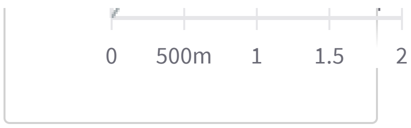
Observation	Evidence from Plot
1. LSTM and GRU outperform RNN	Yellow lines in parallel plot align with LSTM and GRU. Correlation table shows `cell_type` has the highest impact on `val_acc`.
2. Larger hidden dimensions improve accuracy	High-performing runs mostly have `hid_dim ≥ 140`. Sharp drop in performance for `hid_dim = 60`.
3. Dropout helps regularization	Best runs had `dropout` around 0.25–0.35. Correlation table supports positive influence.
4. 2-layer models generalize better	Most high-accuracy runs use `n_layers = 2` as seen in the parallel coordinates plot.
5. Beam size > 1 improves decoding	Models with `beam_size = 2 or 3` generally achieve better validation accuracy.











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)

Here is a precise and markdown-formatted response for **(c) Error Analysis**, based on the provided table from your final predictions:

Without Teacher forcing

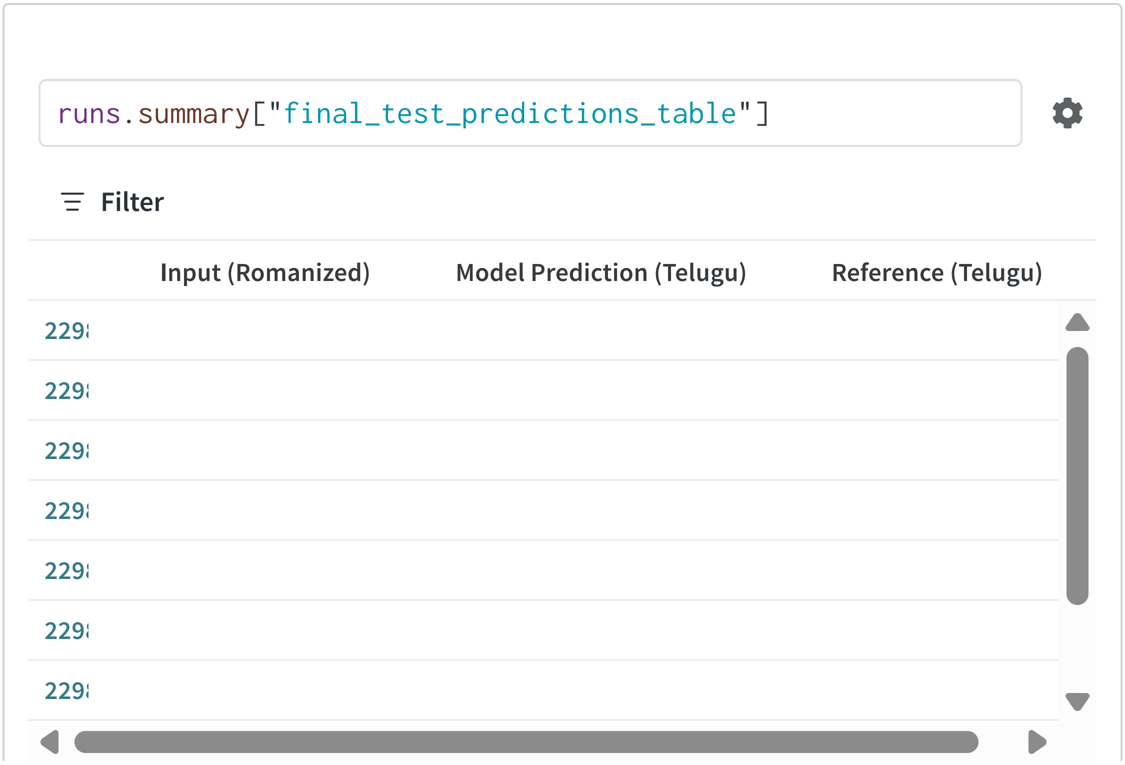
- **Repetitive Predictions (Decoder Looping Error)**
  - The model frequently outputs long, nonsensical repetitive sequences , indicating a failure to stop decoding appropriately.
  - This suggests the decoder may not have learned proper end-of-sequence () prediction.
- **Errors Increase with Input Length**

- Longer Romanized inputs (e.g., ankitamai) lead to more severe prediction issues.
- Likely due to difficulty in maintaining long-range dependencies in the decoder.

With Teacher forcing

- **Fails to Generalize to Unseen/Low-Frequency Words**
  - Some test inputs like ankela, ankelanu are relatively uncommon, and the model overfits to generic patterns.
  - The outputs revert to a repeated default sequence when uncertain.
- **Consonant Cluster Handling is Poor**
  - Words starting with complex consonant sequences like amk, ank, ankit are not well handled.
  - Shows the model's difficulty in handling consonant-heavy transliteration mappings.

After Teacher forcing mechanism the test accuracy is '43%'



```
runs.summary["final_test_predictions_table"]
```

	Input (Romanized)	Model Prediction (Telugu)	Reference (Telugu)
229			
229			
229			
229			
229			
229			
229			

22980 - 22988 of 22988 Export as CSV Columns...

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)

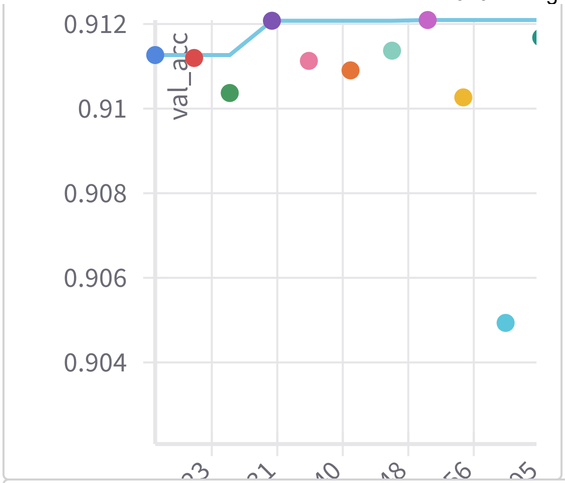
(d) In a 3 x 3 grid paste the attention heatmaps for 10 inputs from your test data (read up on what are attention heatmaps).

After adding attention mechanism the test accuracy increased to 67.44%

val_acc v. created

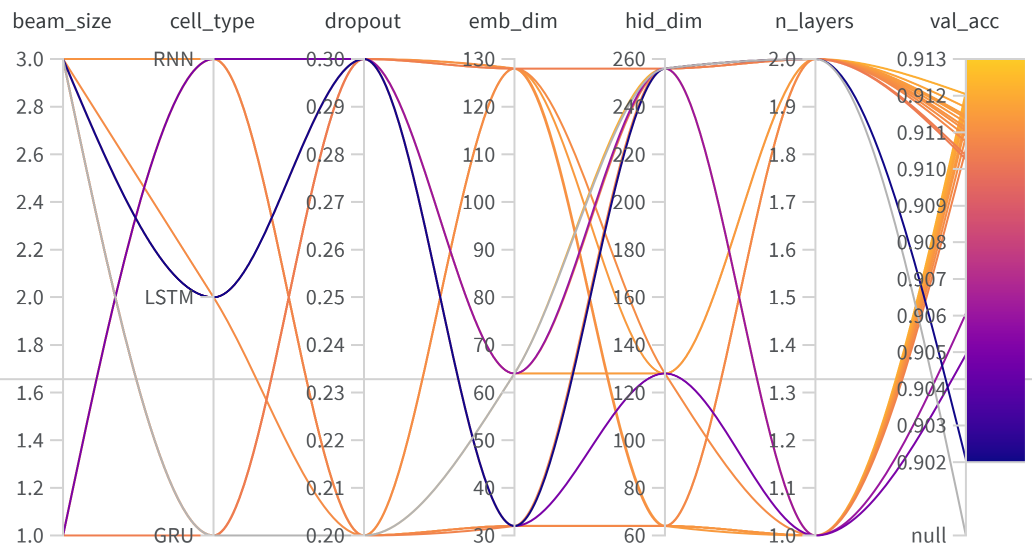
Parameter importance with respect to

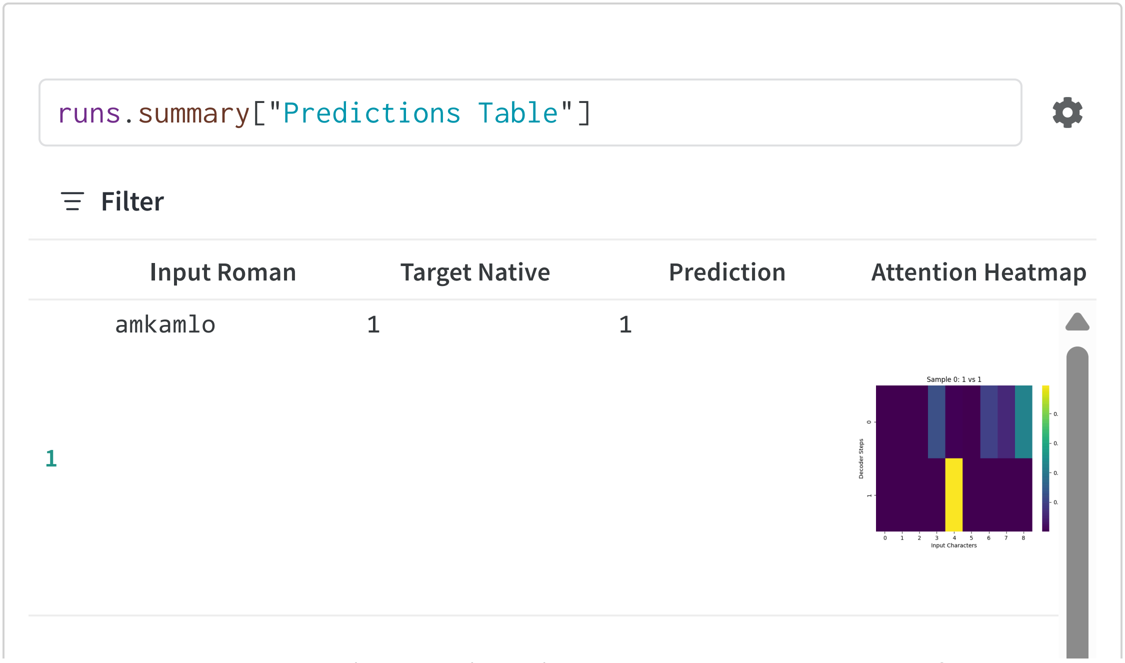
train_acc



Search

Config para…	Importance
hid_dim	
emb_dim	
n_layers	





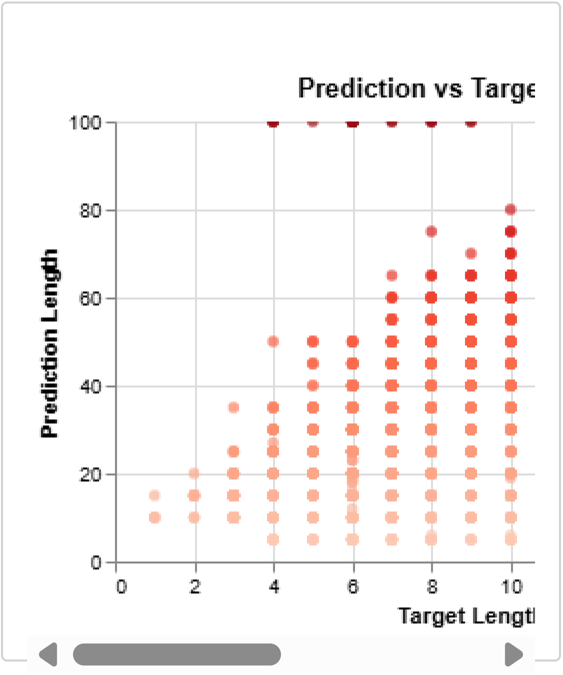
```
runs.summary["Predictions Table"]
```

Filter

	Input Roman	Target Native	Prediction	Attention Heatmap
1	amkamlo	1	1	

2

1 - 2 of 5747 Export as CSV Columns... Res



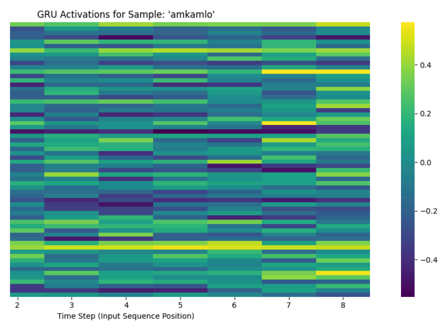
Question 6 (20 Marks)

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!

GRU Activations



Question 7 (10 Marks)

Paste a link to your github code for Part A

Example:
https://github.com/PriyankaDA24M013/DA6401_Assignment-3.git;

- We will check for coding style, clarity in using functions and a README file with clear instructions on training and evaluating the model (the 10 marks will be based on this).
- We will also run a plagiarism check to ensure that the code is not copied (0 marks in the assignment if we find that the code is plagiarised).

- We will check the number of commits made by the two team members and then give marks accordingly. For example, if we see 70% of the commits were made by one team member then that member will get more marks in the assignment (**note that this contribution will decide the marks split for the entire assignment and not just this question**).
- We will also check if the training and test splits have been used properly. You will get 0 marks on the assignment if we find any cheating (e.g., adding test data to training data) to get higher accuracy.

Self Declaration

I, K Priyanka Saraswathi(Roll no: DA24M013), 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.

https://wandb.ai/da24m013-iit-madras-alumni-association/DA6401-A3/reports/DA6401-Assignment-3--VmlldzoxMjgyNTQ3MA