# CS6910 - Assignment 3

Use recurrent neural networks to build a transliteration system.

Rishanth R cs18b044

Number of cheat days used: 4.6 days

Link to this report:
https://wandb.ai/rishanthrajendhran/cs6910_assignment3/reports/CS6910-Assignment-3--Vmlldzo2OTUxMDk

# 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.
- We strongly recommend that you work on this assignment in a team of size 2. Both the members
  of the team are expected to work together (in a subsequent viva both members will be expected to answer questions, explain the code, etc).
- 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 belo using wandb.ai. You can start by cloning this report using the cl 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:

$x$. $\quad y$

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., $V$)

Lets assume a simple RNN cell which does the following computations:

s_$i$ = $f$(U.$x$_i + W.s_i-1 +b)

y_$i$ = $g$(V.s_$i$ + c)

The number of computations for each cell is k + d(2(m+d) + 1) + k(2(k+d) + 1)  #Assume f() and g() counts as one computation

For T such cells:  T(k + d(2(m+d) + 1) + k(2(k+d) + 1))

Encoder and Decoder combined: 2T(k+ d(2(m+d) + 1) + k(2(k+d) + 1))

For softmax: T(k + k) = 2Tk

Total number of computations = 2T(d(2(m+d)+1) + k(2(k+d)+3))

(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., $V$)

For a simple RNN cell: U (m x k), V (k x k), W (k x k) and bias b (k x 1)

Encoder and decoder combined: 2(mk + 2*k^2 + k)

Note: Embedding layer and dense layer have not been included in these calculations as question specifies embedding as the input to the model and does not make a mention of a dense layer after the decoder

# 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)

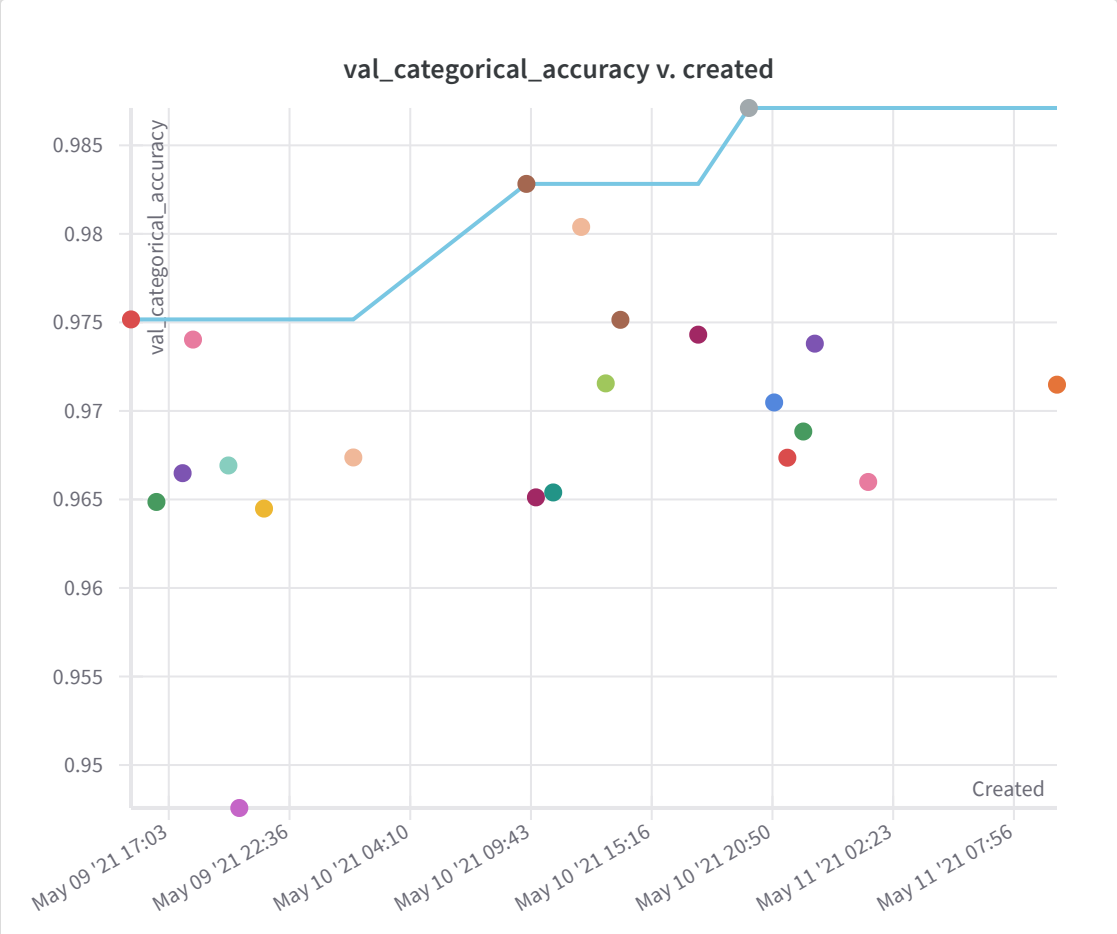
Using the sweep feature in wandb find the best hyperparameter configuration. Here are some suggestions but you are free to decide which hyperparameters you want to explore

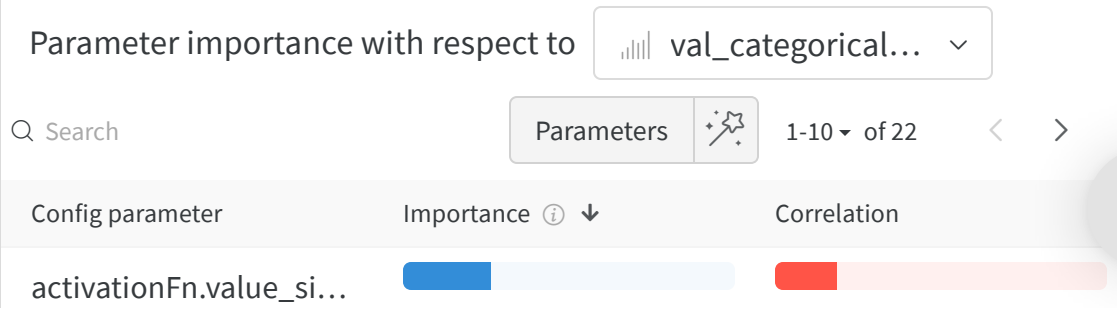
- input embedding size: 16, 32, 64, 256, ...
- number of encoder layers: 1, 2, 3
- number of decoder layers: 1, 2, 3
- hidden layer size: 16, 32, 64, 256, ...
- cell type: RNN, GRU, LSTM
- dropout: 20%, 30% (btw, where will you add dropout? you should read up a bit on this)
- beam search in decoder with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

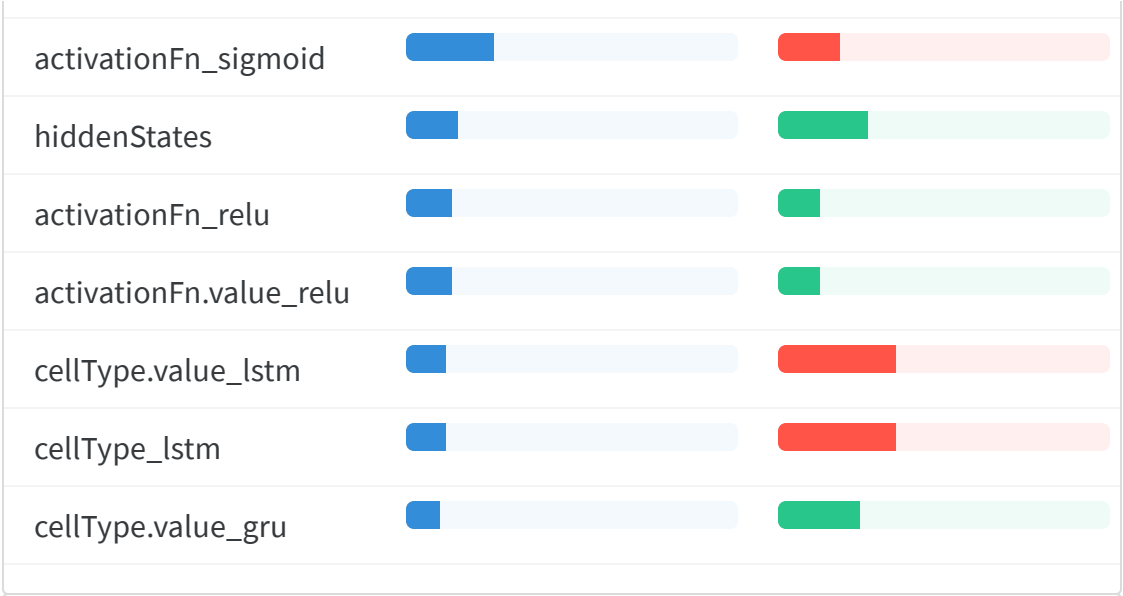
Based on your sweep please paste the following plots which are automatically generated by wandb:

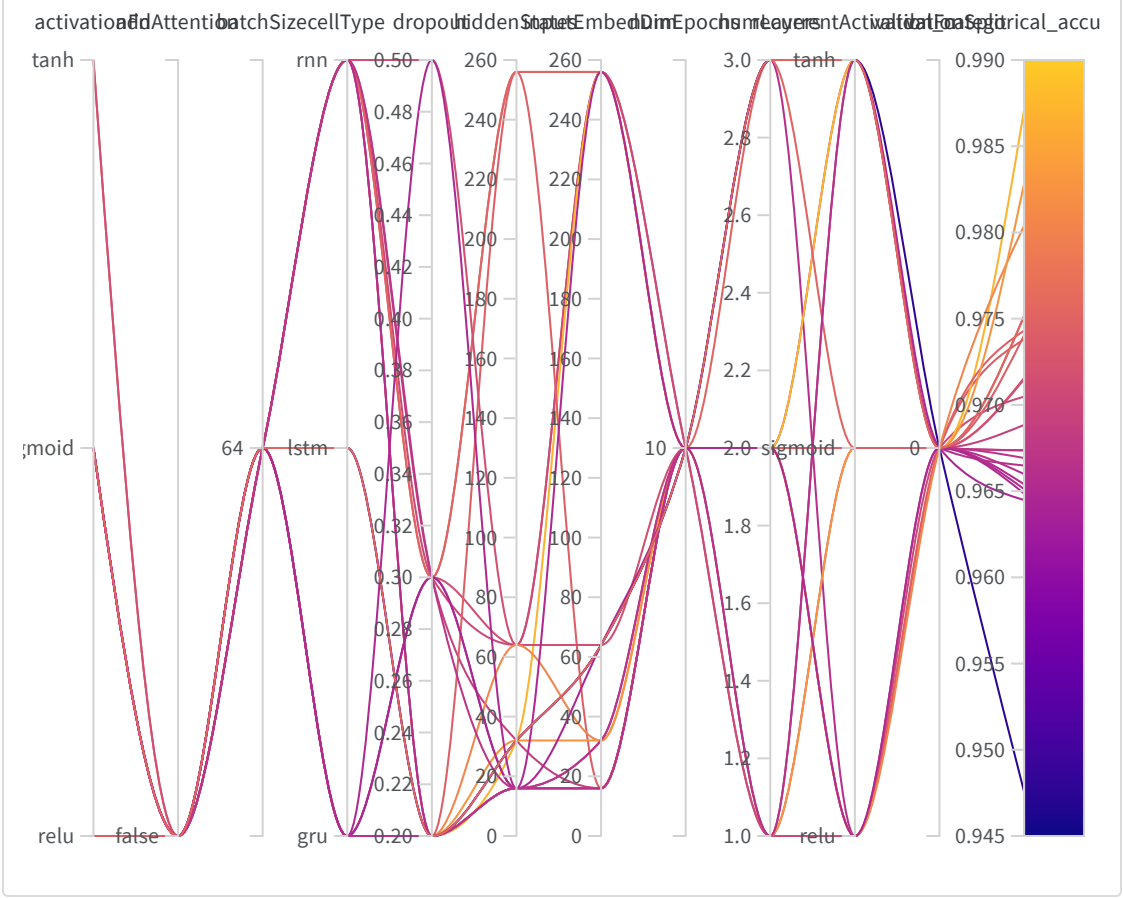
- accuracy v/s created plot (I would like to see the number of experiments you ran to get the best configuration).
- parallel co-ordinates plot
- correlation summary table (to see the correlation of each hyperparameter with the loss/accuracy)

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.






Note: Accuracy mentioned above is character-level accuracy and not word-level accuracy

Language Chosen: Tamil

Hyperparameters choices:

```
inputEmbedDim:
  values: [16, 32, 64, 128, 256]
numLayers:
  values: [1, 2, 3]
hiddenStates:
  values: [16, 32, 64, 256]
cellType:
  values: ["rnn", "gru", "lstm"]
dropout:
  values: [0.2, 0.3, 0.5]
activationFn:
  values: ["tanh", "sigmoid", "relu"]
recurrentActivationFn:
  values: ["tanh", "sigmoid", "relu"]
```

Bayes search strategy was chosen over grid search and random search so as to reduce computational expense and have a better probability of finding the best model in lesser time.

# 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 wrot some random comments that came to my mind)

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

- Models with dropout = 0.2 did much better than higher dropouts and the models with dropout=0.3 did better than models with dropout=0.5 . This could be because higher dropouts led to loss of important feature information which is necessary to make the right predictions. A dropout = 0.2 seems like a sweet spot.
- As the number of hidden states in the encoder and decoder layers were increased, the model performed better. More hidden states would mean that the RNN can model more complex models and hence the better accuracy.
- As the number of encoder-decoder layers were increased, the performance on the validation set took a dip. This might be because of overfitting which could have happened when the number of layers were increased.
- As the size of the input dimensions were increased, the performance improved. This could be because a larger embedding is able to capture more features of the given input thus giving the model more discriminatory power.
- GRU performed much better than LSTMs and simple RNNs keeping other parameters constant. The convergence also happened much sooner in case of GRUs. Compared to LSTMs, RNNs performed better for this task.
- Using relu and tanh activation functions gave better performance than sigmoid function.

# 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).

Due to several mistakes in output dataset, a leeway is given to the model. If the number of corrections to be made to convert the decoder output to the expected output sequence is lesser than 2 and if the length of the decoder output and the expected sequence is at least twice as large as the number of corrections to make, it is considered a partial match.

**Word-level accuracy:** 16.24% (Perfect matches only i.e. all characters matched between prediction and expected output)

**Word-level accuracy including partial matches:** 29.35%

Predictions on test data set available as "predictions_vanilla.txt" in GitHub.

---

(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.

Input sequence: இந்தியாவிலும்

Output sequence:  Indhiyaavilum

Decoded sentence:  indiyavargal

Correct Prediction?: False

Number of corrections required: 9

Comment: As noted before, the first half of the prediction is quite accurate, except for small mistakes such as 'd' in place of 'dh'. The second half of the prediction (last 5 characters) has many mistakes.

---

Input sequence: அவுஸ்திரேலியாவிலும்

Output sequence:  Australiyaavilum

Decoded sentence:  ameruviyarum

Correct Prediction?: False

Number of corrections required: 14

Comment: The model has probably memorised a previous input sequence whose expected output was 'Americavilum' and has reproduced that output with small modifications after seeing the same staring 'A'.

---

Input sequence: வாழ்ந்தவர்

Output sequence:  vaazhnthavar

Decoded sentence:  vaalaththaana

Correct Prediction?: False

Number of corrections required: 9

Comment: The model mispredicts 'zha' as 'la', 'in' as 'th', and 'var' as 'ana'. I suspect that the model would have encountered the word 'valandha' before, and because of its similarity, has predicted a similar output here.

---

Input sequence: டாப்ளர்

Output sequence:  Topler

Decoded sentence:  taalar

Correct Prediction?: False

Number of corrections required: 6

Comment: The model could not have predicted this unless the proper noun 'Topler' was present in the train set. The Tamil spelling distorts the English pronunciation of the name. The predicted 'taalar' is close to the Tamil pronunciation 'Taaplar'.

---

Input sequence: தனது

Output sequence:  thanathu

Decoded sentence:  thanathu

Correct Prediction?: True

No corrections required!

---

Input sequence: பெயரை

Output sequence:  peyarai

Decoded sentence:  periya

Correct Prediction?: False

Number of corrections required: 5

Comment: The model mixes up the order of the 'ya' and 'ra' sounds in this example.

---

Input sequence: கிறிஸ்டியன்

Output sequence:  Christian

Decoded sentence:  kiriyaan

Correct Prediction?: False

Number of corrections required: 9

Comment: The model could not have predicted the expected output as the Tamil spelling distorts the English pronunciation. The predicted 'Kiriyaan' is somewhat similar to the Tamil pronunciation 'Kiristiyan'.

---

Input sequence: டாப்ளர்

Output sequence:  Topler

Decoded sentence:  taalar

Correct Prediction?: False

Number of corrections required: 6

---

Input sequence: எனக்

Output sequence:  enak

Decoded sentence:  yenda

Correct Prediction?: False

Number of corrections required: 3

Comment: The model is confused between the vowel 'எ' and the consonant 'ய'.

---

Input sequence: கூறிக்கொண்டார்

Output sequence:  koorikkondaar

Decoded sentence:  koorikkappatta

Correct Prediction?: False

Number of corrections required: 9

Comment: As noted before, the model does well in the first half of the output sequence but falters in the second half. I suspect that the model had encountered the word 'kurikkapatta' before which is spelt similar to 'koorikkondar'.

---

(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!
- ...

Despite a very high character-level accuracy on train data set, this model performed badly based on word-level accuracy on test data set.

This model performed fine on short input sequences. In case of long input sequences, it predicted the first half of the output quite accurately. It faltered in predicting the second half when the expected output was a long sequence.

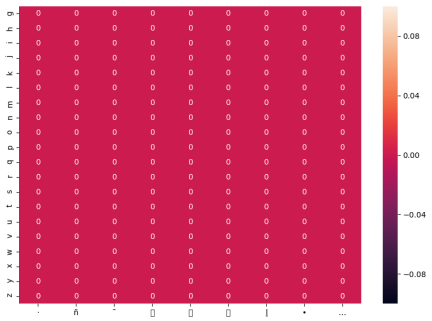
Due to a large number of unique tokens, plotting the confusion matrix at one go was difficult. It had to be split up into smaller matrices and then plotted.

It is observed that around 25k words required 5 or lesser number of corrections. Another 20k required between 6 and 10 corrections and very few required more than 10 corrections.

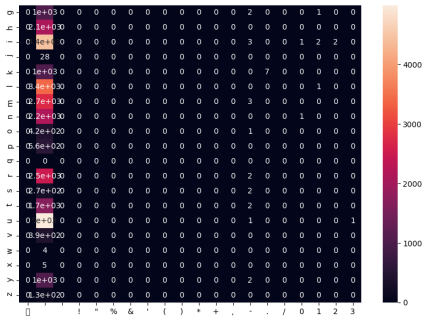
In the confusion matrix plot*2*_2, we observe a nearly diagonal relationship between the prediction and expected output which would mean that the model did a fair job in predicting the characters 'g' to 'z'.

Plot1_1 shows that the character 'a' was rightly predicted a large number of times.

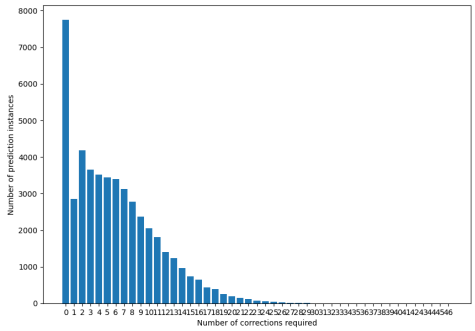
plot_2_3



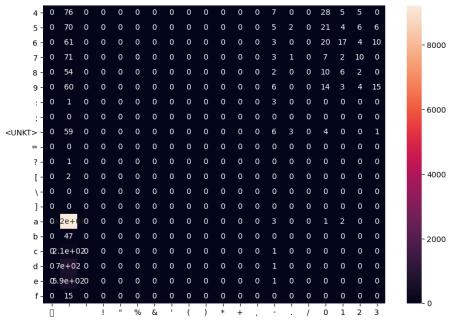
plot_2_0



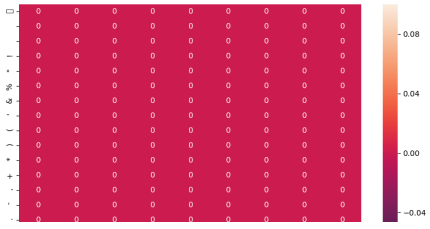
## plotCorrectionFrequency

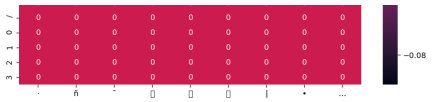



## plot_1_0

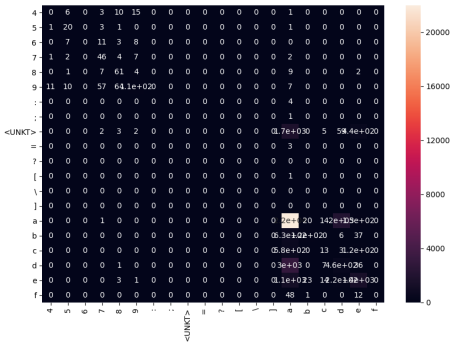


## plot_0_3

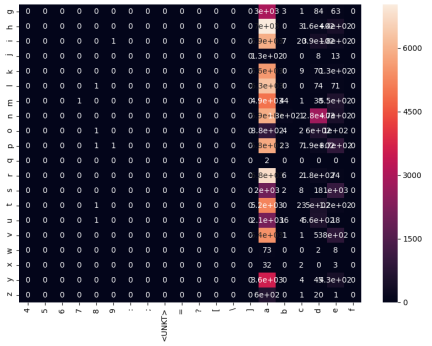




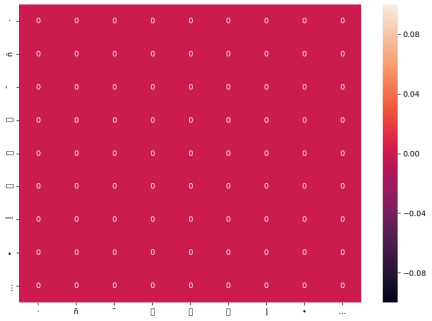
## plot_1_1



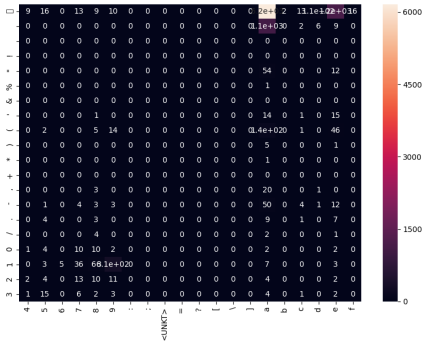
## plot_2_1



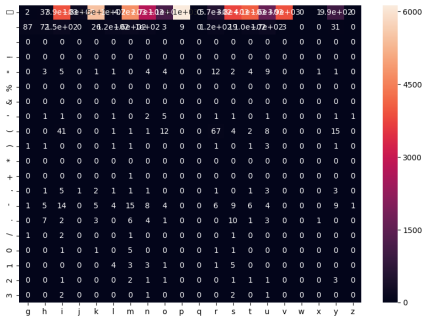
## plot_3_3



## plot_0_1

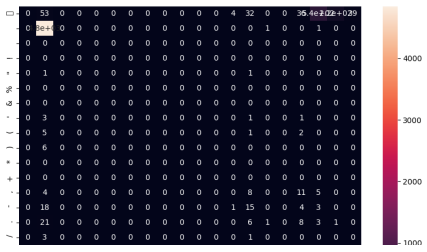


## plot_0_2

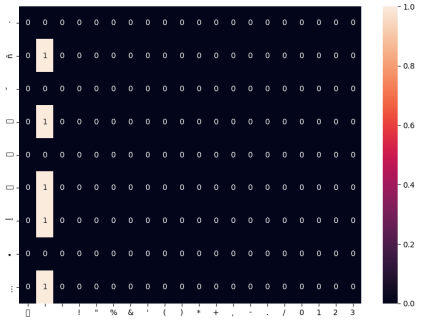


## plot_2_2

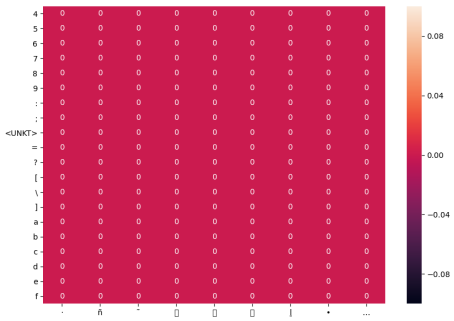
## plot_0_0



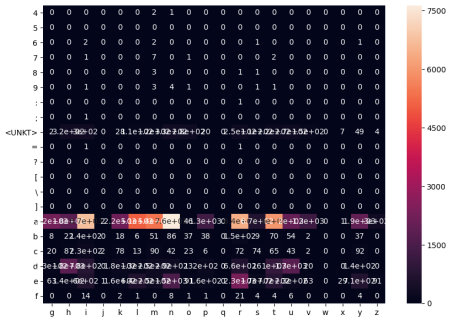
## plot_3_0



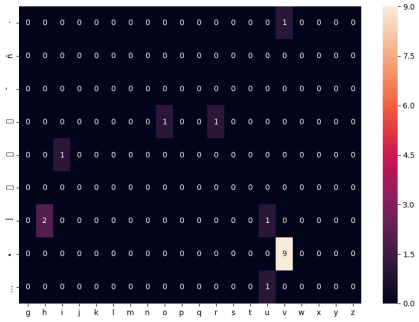
## plot_1_3



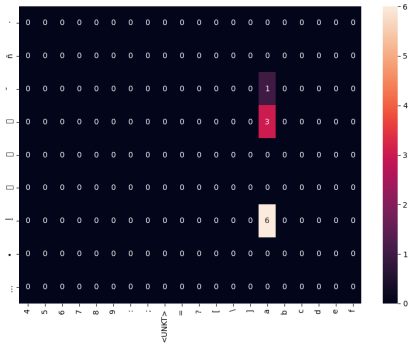
## plot_1_2



## plot_3_2



plot_3_1

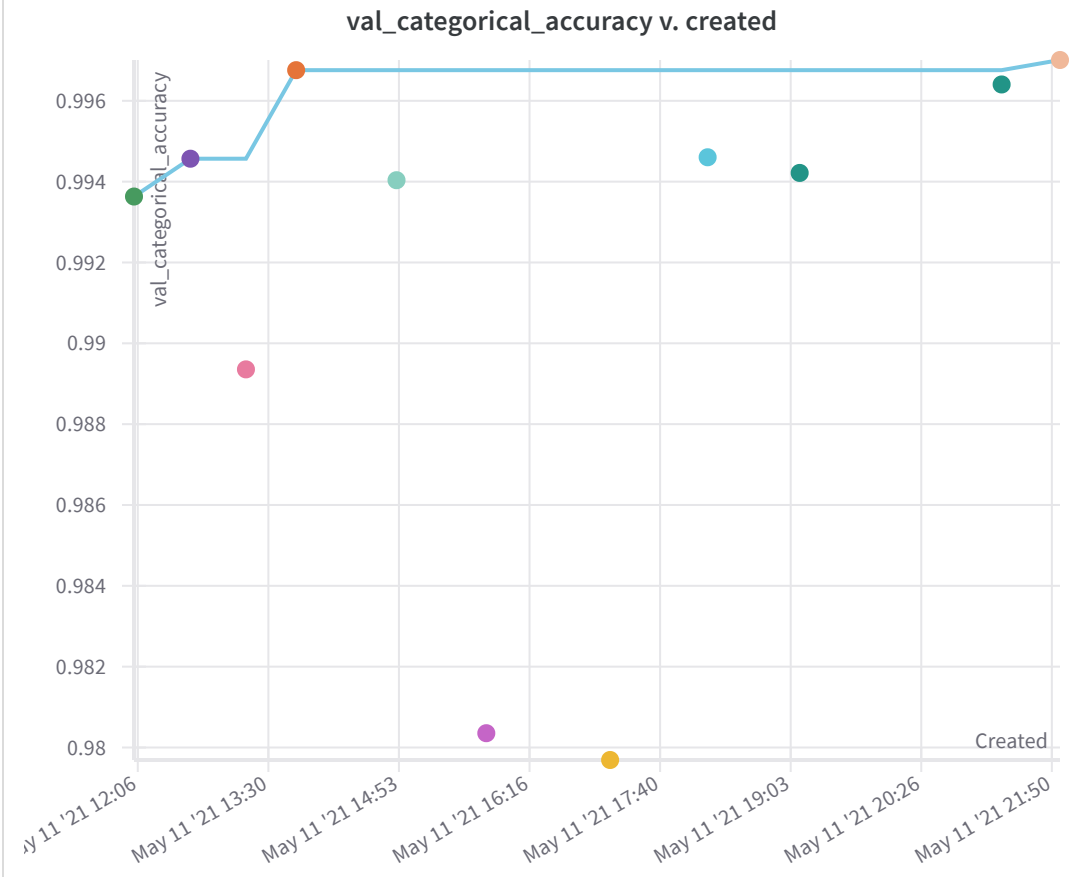


# Question 5 (20 Marks)

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.

Hyperparameters were tuned again for attention model. Number of encoder-decoder layers was fixed at 1.




Parameter importance with respect to val_categorical…

Search

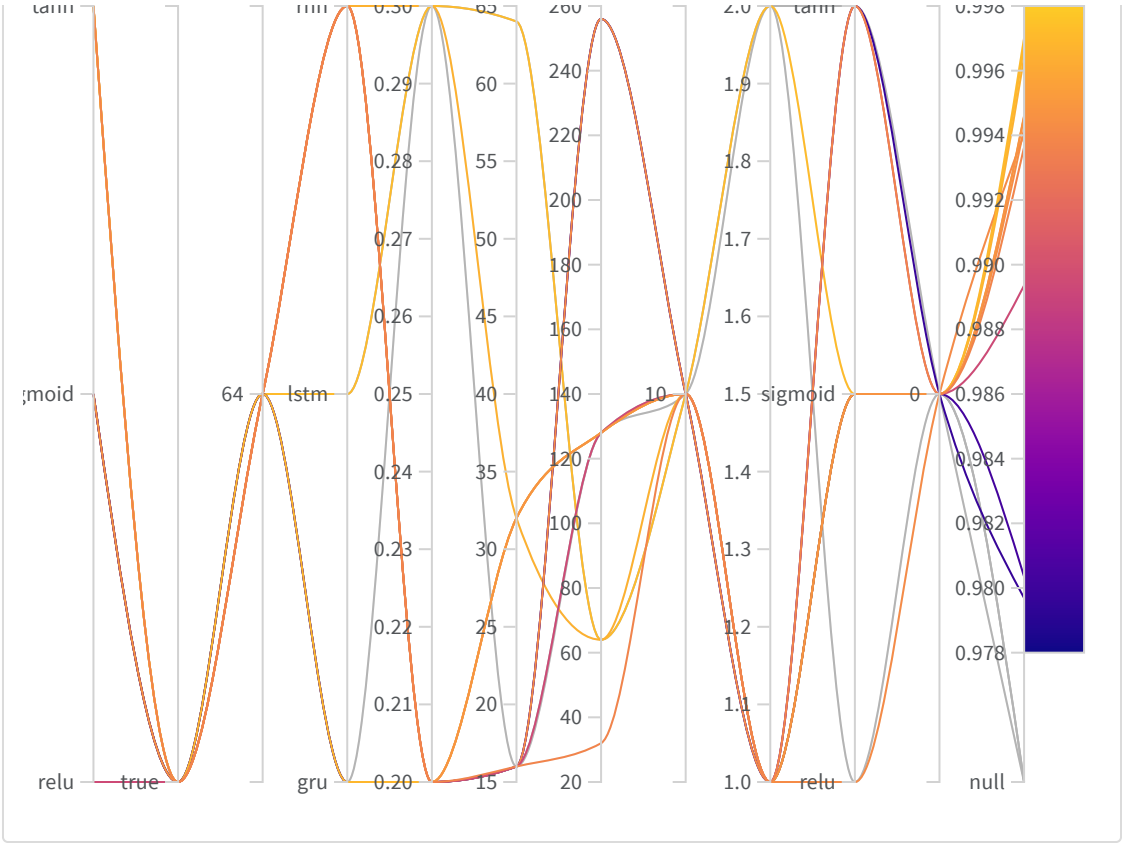
Parameters

1-10 of 13

| Config parameter | Importance | Correlation |
|---|---|---|
| inputEmbedDim | | |
| cellType.value_gru | | |
| hiddenStates | | |
| activationFn.value_tanh | | |
| activationFn.value_si… | | |
| activationFn.value_relu | | |
| cellType.value_rnn | | |
| recurrentActivationFn… | | |

activationFn batchSize cellType dropout hiddenStates inputEmbedDim Epochs numLayers recurrentActivationFn validation_categorical_accu

tanh rnn 0.20 65 360 2.0 tanh 0.998




Note: Accuracy mentioned above is character-level accuracy and not word-level accuracy

Language Chosen: Tamil

Hyperparameters choices:

inputEmbedDim:

values: [16, 32, 64, 128, 256]

hiddenStates:

values: [16, 32, 64, 256]

cellType:

values: ["rnn", "gru", "lstm"]

dropout:

values: [0.2, 0.3, 0.5]

activationFn:

values: ["tanh", "sigmoid", "relu"]

recurrentActivationFn:

values: ["tanh", "sigmoid", "relu"]

Bayes search strategy was chosen over grid search and random search so as to reduce computational expense and have a better probability of finding the best model in lesser time.

---

(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.

**Best Model:**

**Hyperparameters:**

inputEmbedDim = 64,

numLayers = 1,

hiddenStates = 64,

cellType = "lstm",

dropout = 0.3,

activationFn = "tanh",

recurrentActivationFn = "sigmoid"

Due to several mistakes in output dataset, a leeway is given to the model. If the number of corrections to be made to convert the decoder output to the expected output sequence is lesser than 2 and if the length of the decoder output and the expected sequence is at least twice as large as the number of corrections to make, it is considered a partial match.

**Word-level accuracy:** 54.55% (Perfect matches only i.e. all characters matched between prediction and expected output)

**Word-level accuracy including partial matches:** 82.73%

Predictions on test data set available as "predictions_attention.txt" in GitHub.

---

(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)

The attention model performed very well even on long input sequences, which was a weak area for the non-attention based model. Most of the predictions categorised as wrong on the test data set involved small phonetic mistakes which is due to the inherent ambiguity in Tamil.

E.g. Tamil only has a single letter 'க' sounds such as ka, kha, ga, gha (Devanagari-equivalent sounds)

i.e. Tamil does not have distinct aspirated consonants which makes transliteration difficult.

Input sequence: இந்தியாவிலும்

Output sequence:  Indhiyaavilum

Decoded sentence:  indhiyaavilum

Correct Prediction?: True

No corrections required!

---

Input sequence: அவுஸ்திரேலியாவிலும்

Output sequence:  Australiyaavilum

Decoded sentence:  avustreliyaavilum

Correct Prediction?: False

Number of corrections required: 3

Comment: The model could not have predicted the expected output unless it had already seen it because the Tamil spelling distorts the English pronunciation. The pronunciation of the Tamil spelling is 'Avusthireliyavilum' which is pretty close to what the model has predicted.

---

Input sequence: வாழ்ந்தவர்

Output sequence:  vaazhnthavar

Decoded sentence:  vaazhnthavar

Correct Prediction?: True

No corrections required!

---

Input sequence: டாப்ளர்

Output sequence:  Topler

Decoded sentence:  dapler

Correct Prediction?: False

Comment: As mentioned for the vanilla model, this model has predicted the pronunciation of the Tamil spelling as it is. Another interesting observation is that the attention model predicts 'டா' to be 'da' whereas the non-attention model predicts it as 'ta'. It is is important to note that Tamil lacks aspirated consonants and the model's confusion regarding the pronunciation stems from the fact that the same word 'ட' can be used for both the 'ta' and 'da' sounds in Tamil.

---

Input sequence: தனது

Output sequence:  thanathu

Decoded sentence:  thanathu

Correct Prediction?: True

No corrections required!

---

Input sequence: பெயரை

Output sequence:  peyarai

Decoded sentence:  peyarai

Correct Prediction?: True

No corrections required!

---

Input sequence: கிறிஸ்டியன்

Output sequence:  Christian

Decoded sentence:  kiristiyan

Correct Prediction?: False

Comment: The model could not have predicted the expected output as the Tamil spelling distorts the English pronunciation. The predicted 'Kiristiyan' is exactly the Tamil pronunciation.

---

Input sequence: டாப்ளர்

Output sequence:  Topler

Decoded sentence:  dapler

Correct Prediction?: False

---

Input sequence: எனக்

Output sequence:  enak

Decoded sentence:  enak

Correct Prediction?: True

No corrections required!

Comment: The attention model does not confuse the vowel 'எ' and the consonant 'ஷ'.

---

Input sequence: கூறிக்கொண்டார்

Output sequence:  koorikkondaar

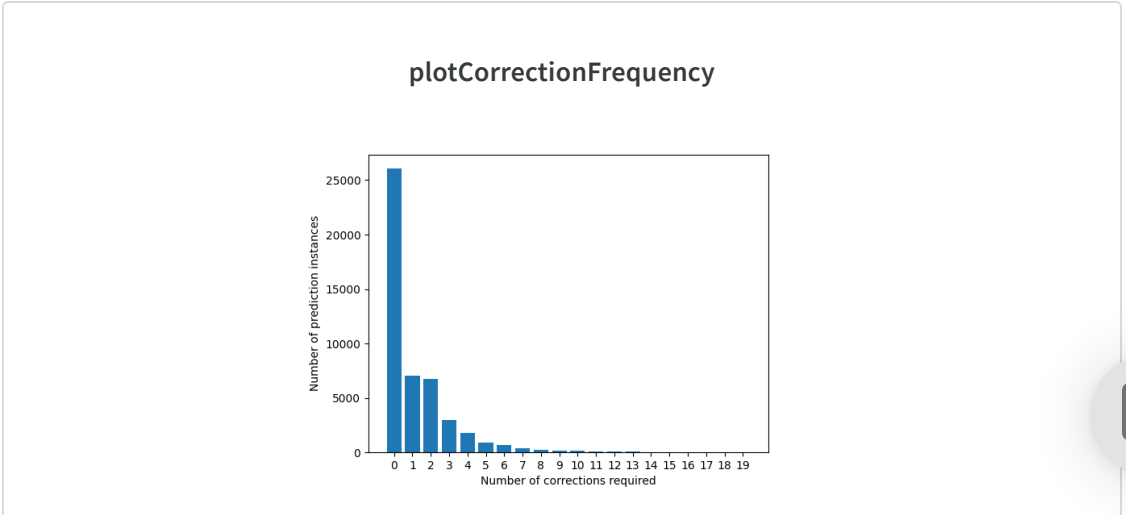
Decoded sentence:  koorikkondaar

Correct Prediction?: True

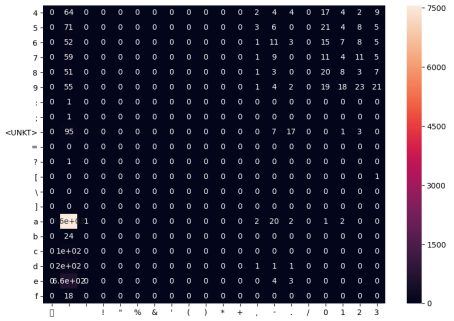
No corrections required!

Clearly, the attention model is performing much better than the non-attention model, not just on long sequences but also on short sequences. The attention model learns to focus on the right characters in the input sequence to predict a character in the output sequence which is reflected in the heat maps visualised below.

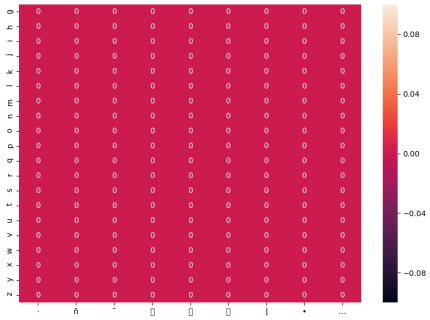
Around 44k of the total 48k predictions required less than 5 corrections and the max number of corrections required was 13. Clearly the attention model has done so much better than the non-attention model which needed 29 corrections at maximum and had only 25k words under the lesser-than-5-corrections bracket.



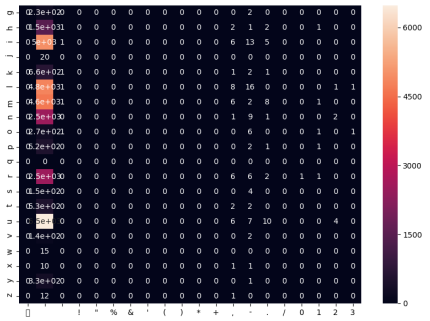
## plot_1_0



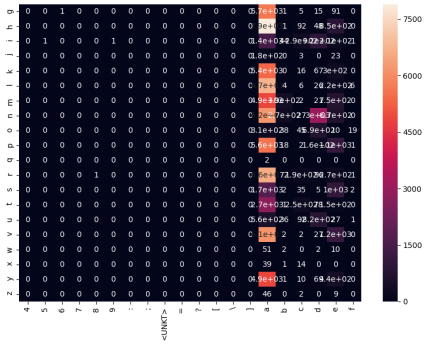
## plot_2_3



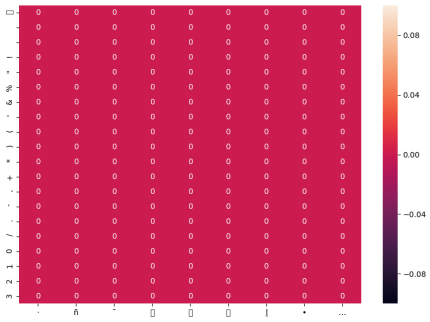
## plot_2_0



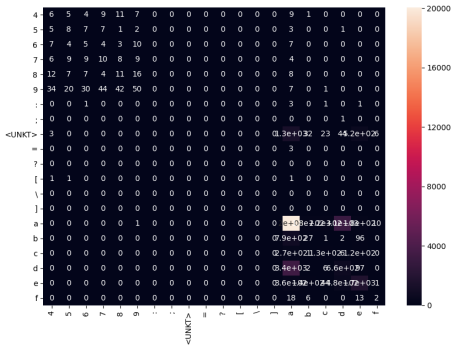
## plot_2_1



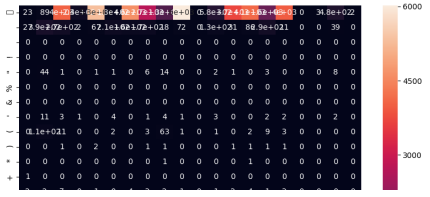
## plot_0_3

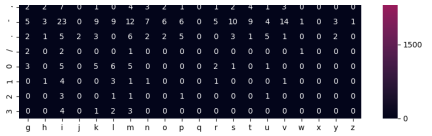


## plot_1_1

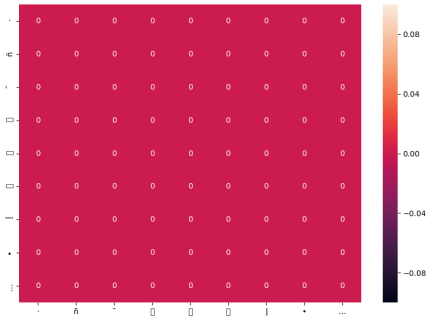


## plot_0_2

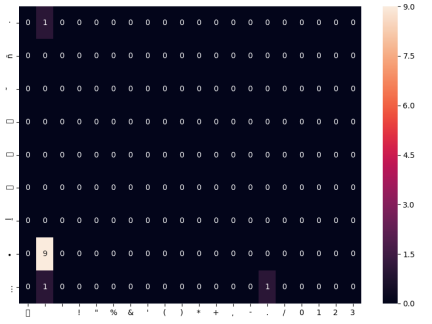




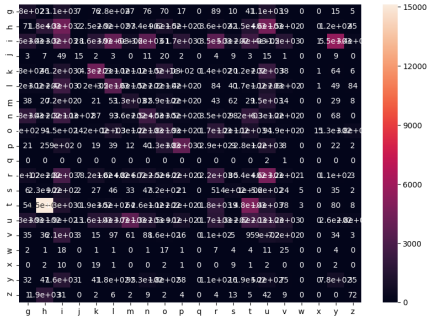
## plot_3_3



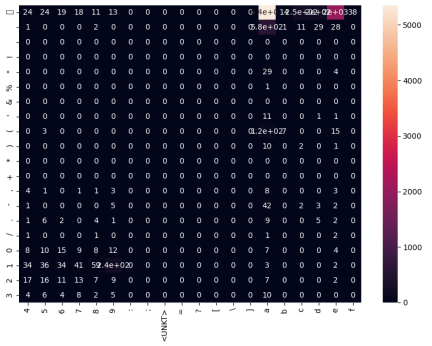
## plot_3_0



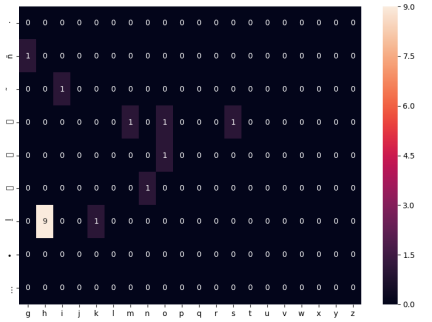
## plot_2_2



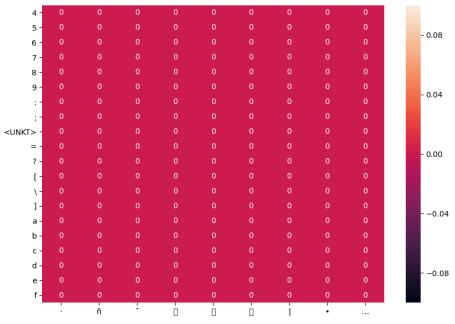
## plot_0_1



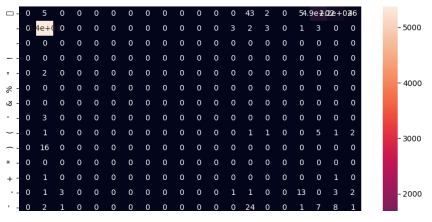
## plot_3_2

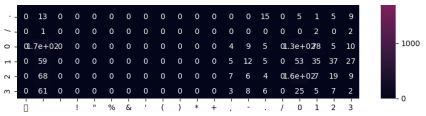


## plot_1_3

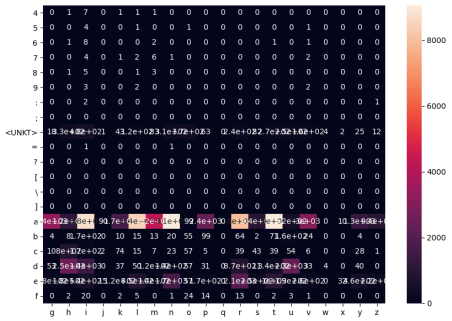


## plot_0_0

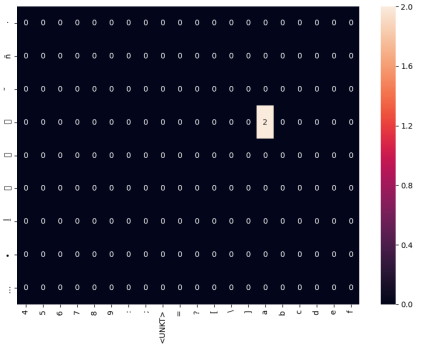




plot_1_2

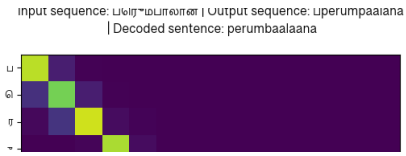


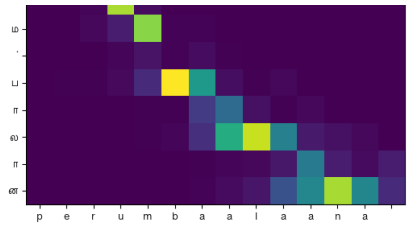
plot_3_1



(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).

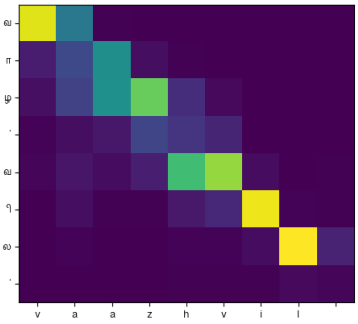
plot_8





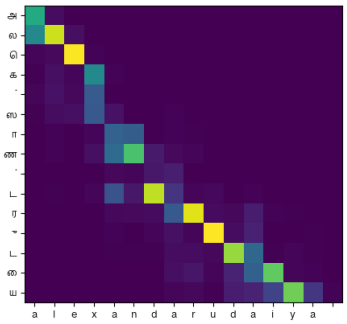

## plot_7

Input sequence: வாழ்வில் | Output sequence: ▯vaazhvil
| Decoded sentence: vaazhvil



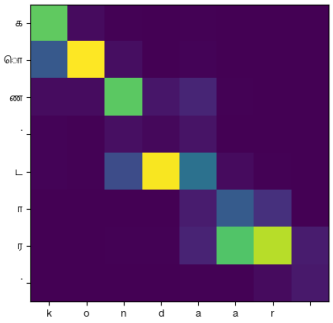

## plot_6

Input sequence: அலெக்ஸாண்ட்ருடைய | Output sequence: ▯Alexandarudaiya
| Decoded sentence: alexandarudaiya



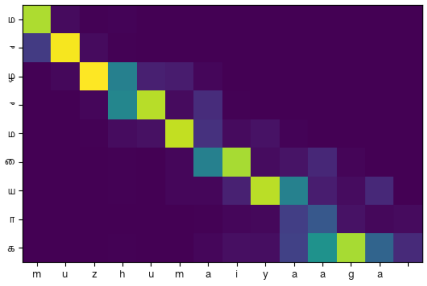

## plot_5

Input sequence: கொண்டார் | Output sequence: ▯kondaar
| Decoded sentence: kondaar



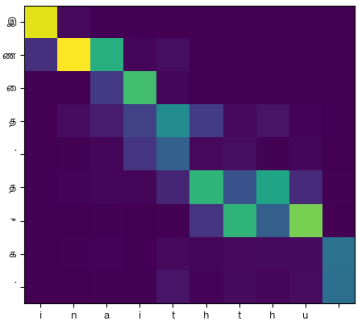
## plot_3

Input sequence: முழுமையாக | Output sequence: □mulumaiyaaga
| Decoded sentence: muzhumaiyaaga



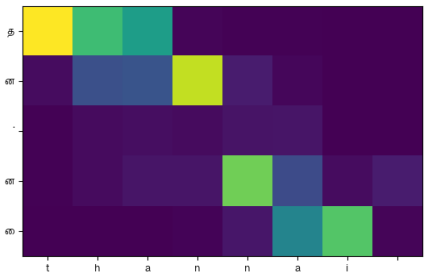

## plot_4

Input sequence: இணைத்தக் | Output sequence: □yinaiththu
| Decoded sentence: inaiththu



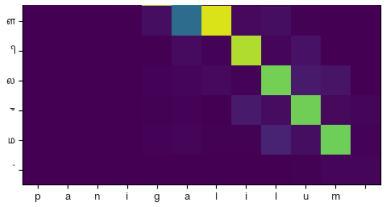

## plot_2

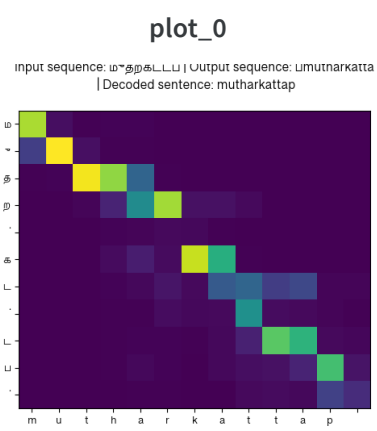
Input sequence: தன்னை | Output sequence: □thannai
| Decoded sentence: thannai

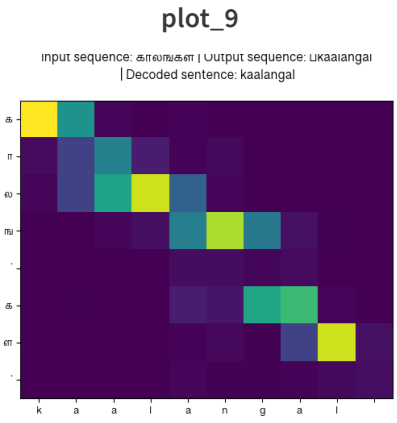



## plot_1

Input sequence: பணிகளிலும் | Output sequence: □panigalilum
| Decoded sentence: panigalilum




# 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!

**Sample Visualisations:**

**Text Attention Visualisation:**

https://github.com/RishanthRajendhran/cs6910_assignment3/blob/main/textAttentionResult.html

**Text Confidence Visualisation:**
https://github.com/RishanthRajendhran/cs6910_assignment3/blob/main/textConfidenceResult.html

Read the GitHub README to learn how to generate these files.

# Question 7 (10 Marks)

Paste a link to your github code for Part A

Github Link:
https://github.com/RishanthRajendhran/cs6910_assignment3

- 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

Team-member : Contribution

Rishanth .R (CS18B044) : 100% contribution

I, Rishanth .R (CS18B044), swear on my honour that the above declaration is correct.