DA6401 – Assignment – 03 (AE21B105)

Student Details:

Name : Joel J
Roll No : AE21B105

WandB link : https://api.wandb.ai/links/A3_DA6401_DL/t5ookbc5
GitHUB link : https://github.com/AE21B105-JoelJ/DA6401_A03.git

DA6401 - Assignment 03

Use Recurrent Neural Networks and Attention mechanism for transliteration task...

Joel_J_ae21b105

Created on May 19 | 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)

(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)

SOL:

The RNN Seq2Seq model is built with the following configurable parameters,

- Input layer size for character embedding
- The hidden state size of the encoder and decoders
- Type of the cell for the encoder and decoder separately
- Number of layers in the encoder and decoder
- Whether the encoder is bi-directional
- Dropout in the encoder and decoder

(a) The total number of computations (Assuming a simple RNN)

Lets say that the initial characters of the sequence are one hot vectors, (computation are similar to Big Oh notation)

- Input : The conversion of one hot encoding to input embedding happens for all time steps in the input sequence. Thus the total computations are $T \times m \times V$
- Encoder : The input embedding for each time step is multiplied by U_{enc} and then the states is computed combining this and the previous states by a W_{enc} matrix and carried on for all time steps, thus the total computations are calculated, [$s_t = \sigma($ $U_{enc}x + W_{enc}s_{t-1} + b)$ where x is m-dimensional and s is k-dimensional], $T \times [k \times m + k \times k + k]$ where the first two are matrix multiplications and the third term is the non linear activation.
- Decoder : In decoder the inputs passed are one hot vectors which are typically the outputs of the previous predictions. Thus the similar computations as the encoder but an addition of computation due to the prediction (projection on the vocabulary with softmax), the total computations are $T \times [k \times V + k \times k + k + V \times k + V]$ where the fourth term is projection to the vocabulary and the fifth term is the non-linear activation that is the softmax here

Total = $(T \times m \times V) + T \times [k \times m + k \times k + k] + T \times [k \times V + k \times k + k + V \times k + V]$

Additional computations if a LSTM or GRU is used is the gate multiplications its of order like $T \times 3 \times [d]$ the 3 comes from the three gates present.

(b) Total number of parameters

Lets assume a RNN,

- Input : The embedding network is the input which takes V dimensional input and gives k dimensional vectors thus the number of parameters is $m \times V + m$ and the same is used across all time steps. Assuming bias is present.
- Encoder : The encoder consists of the U_{enc}, W_{env} and b as mentioned as in the previous question. Thus the total number of parameters is $k \times m + k \times k + k$
- Decoder : The decoder consists similar to the encoder but the extra projector to the vocabulary to the output and also the inputs are one hot vectors of dimension V. Thus the total number of parameters are $(k \times V + k \times k + k) + (V \times k + V)$

Total number of parameters are $(m \times V + m) + (k \times m + k \times k + k) + (k \times V + k \times k + k) + (V \times k + V)$

If we use a LSTM instead of an RNN then the additional parameters would be for the gates U_i, W_i, U_o, W_o, U_f, W_f and biases, U's are dimensions of kxm and W's are dimension of kxk

thus the additional parameters are $3\ (k \times m + k) + 3\ (k \times k + k)$

If we use a GRU instead of an RNN then the additional parameters would be for the gates U_i, W_i, U_f, W_f and biases, U's are dimensions of kxm and W's are dimension of kxk

thus the additional parameters are $2\ (k \times m + k) + 2\ (k \times k + k)$

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:

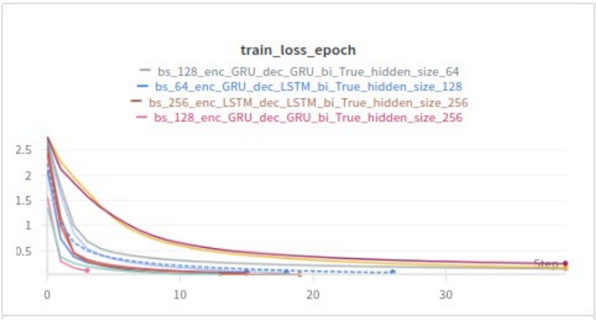
- accuracy v/s created plot (I would like to see the number of experiments you ran to get the best configuration).
- parallel co-ordinates plot
- correlation summary table (to see the correlation of each hyperparameter with the loss/accuracy)

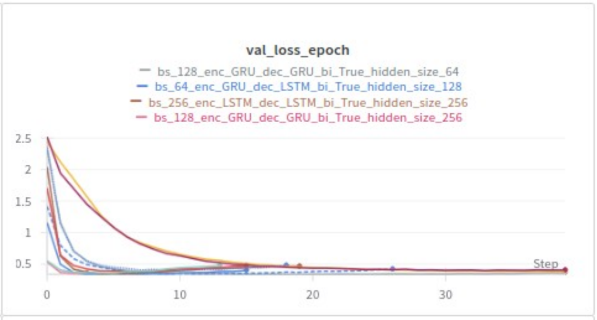
Also write down the hyperparameters and their values that you sweeped over. Smart strategies to reduce the number of runs while still achieving a high accuracy would be appreciated. Write down any unique strategy that you tried for efficiently searching the hyperparameters.

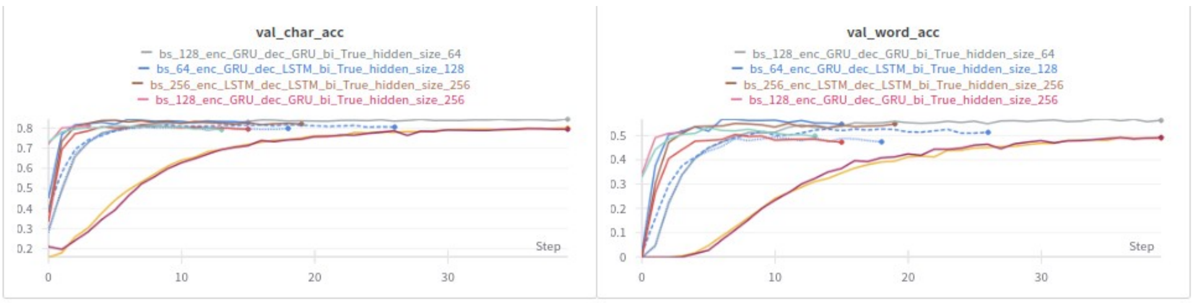
SOL :

Since the RNN takes some time to train and the parameter space to sweep over was large, I followed the following strategy that is to have a large parameter space and run a random sweep for quite some time and after enough number of runs note down the inferences and then reduce the parameter space based on the inference and then run a bayes sweep on top of that to find the best parameters of the model. The sweep configurations used, inferences and plots are attached below
Note : There are two metrics that i have used word-wise accuracy and character-wise accuracy the sweep uses character-wise accuracy to maximize but i have logged the word-wise accuracy as well which will be seen in plots. Also the max epochs were 40 and incorporated early-stopping with a patience parameter of 7 so which prunes runs which dint improve over 7 epochs. Also note that teacher forcing method was used in training decoder and greedy search was used while calculation accuracy.

```
sweep_config = {
    "name" : "Vanilla Seq2Seq",
    "method" : "random",
    "metric" : {
        "name" : "val_char_acc",
        "goal" : "maximize"
    },
    "parameters" : {
        "learning_rate" : {"values" : [0.001]}, # 5 times when using SGD
        "optimizer" : {"values" : ["Adam", "SGD"]},
        "dropout_rnn" : {"values" : [0, 0.1, 0.2, 0.4]},
        "batch_size" : {"values" :  [64, 128, 256]},
        "epochs" : {"values" : [40]},
        "embedding_dim" : {"values" : [16, 32, 64, 128, 256]}, # Embedding size for the input layer
        "num_layers" : {"values" : [1, 2, 3, 5]}, # Number of layers
        "hidden_size_enc" : {"values" : [32, 64, 128, 256]}, # Hidden state size
        "enc_cell_type" : {"values" : ["RNN", "LSTM", "GRU"]},
        "dec_cell_type" : {"values" : ["RNN", "LSTM", "GRU"]},
        "bi_directional" : {"values" : [True, False]}
    }
}
```



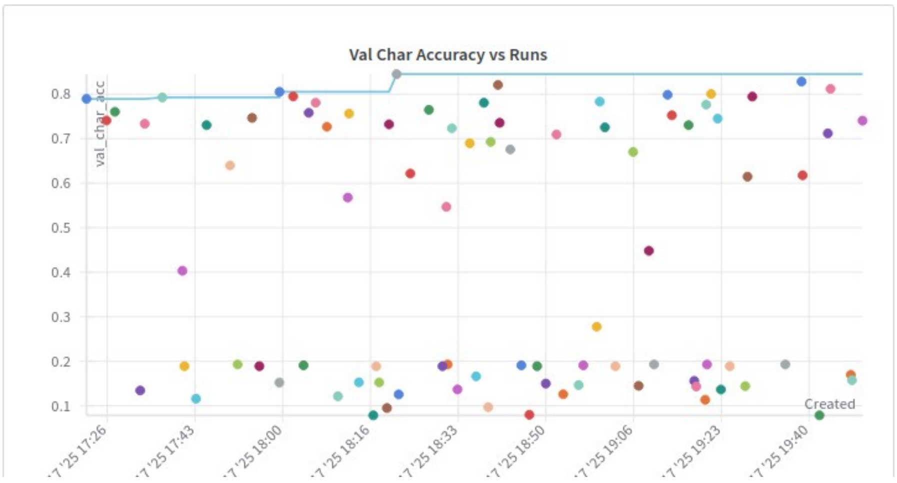


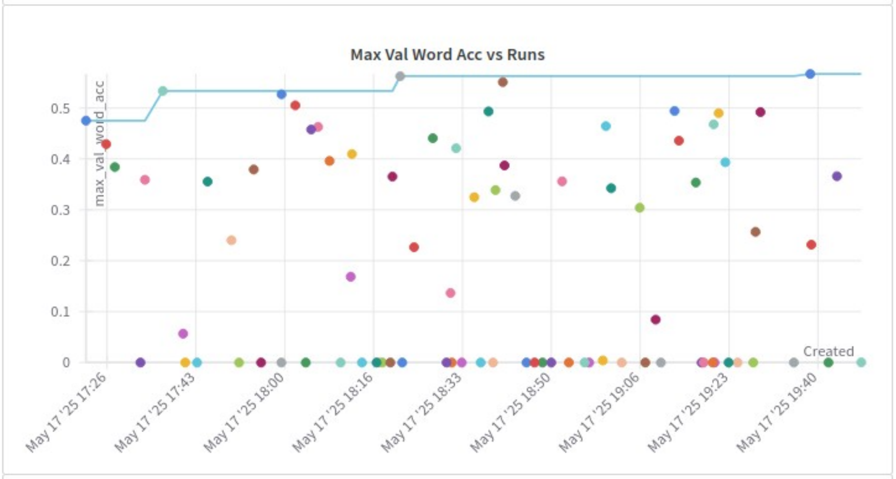


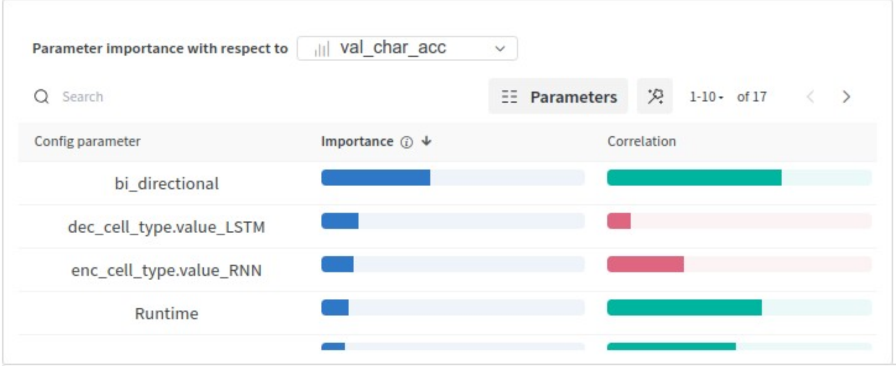
Sweep: Vanilla Seq2Seq 1 89

Sweep: Vanilla Seq2Seq 2 0

The accuracies plot and other plots are attached below,

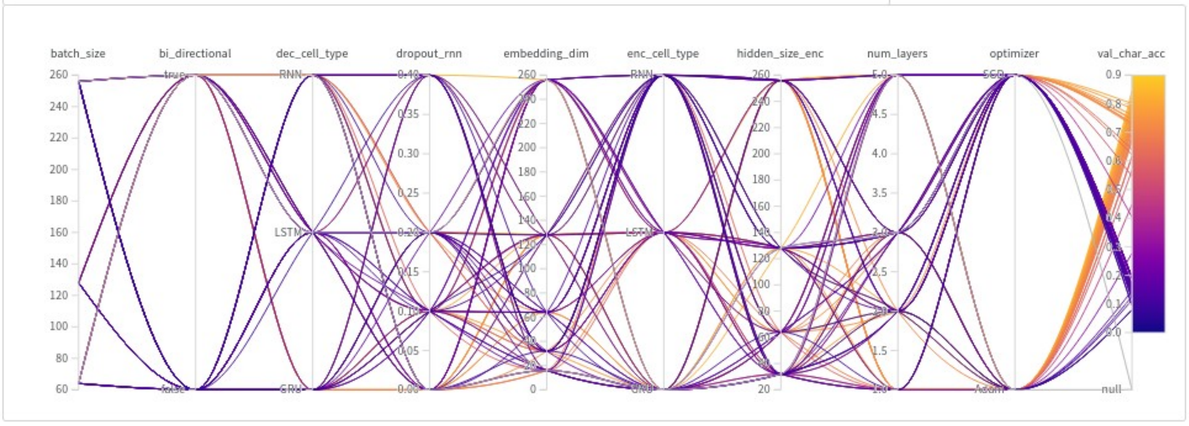





Parameter importance with respect to val_char_acc

Search | Parameters | 1-10 of 17

Config parameter	Importance	Correlation
bi_directional		
dec_cell_type.value_LSTM		
enc_cell_type.value_RNN		
Runtime		



Sweep: Vanilla Seq2Seq 1 89

Sweep: Vanilla Seq2Seq 2 0

The inferences and the maximum accuracy attained are as follows,

Best Model (yet!)

Train loss : 0.1452
Val loss : 0.37036
Val char accuracy : 0.845
Val word accuracy : 0.5625

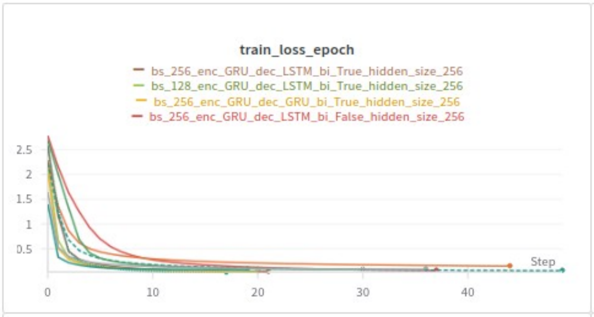
Inferences

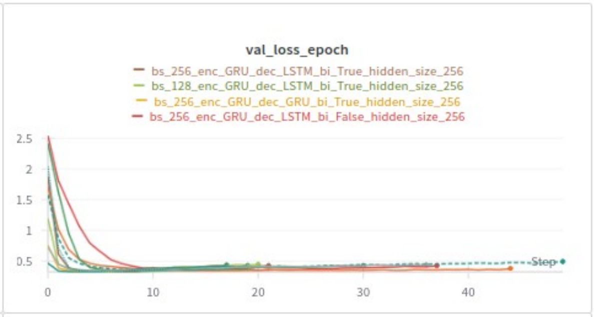
- From the correlation plot we could directly see that for non attention encoder-decoder models bidirectional RNN helps a lot. As also seen from the top 10 best from this sweep, eight of them are bidirectional.
- SGD optimizer was bad and was removed in the next following sweep. Most of the runs with SGD dint train well. On the other hand Adam optimizer was good and started to learn quickly.

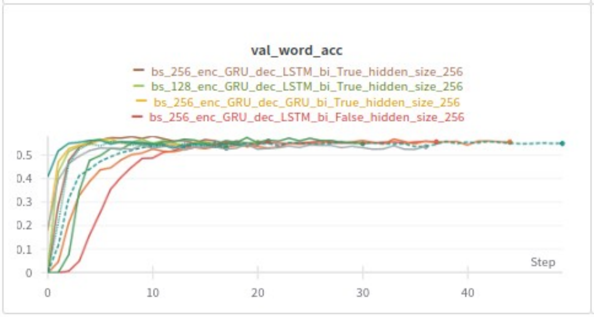
- Encoder RNN was bad when compared to LSTM and GRU, decoder having RNN was somewhat okay but not the best. The top 5 best models of this sweep did not have RNN as either of the decoder or encoder. Thus RNN was removed.
- Having hidden size (hidden state dimension) slightly larger is preferred when compared to the smaller dimension.
- The embedding dimension for input layer is not of very much importance from the covariance plot but from the correlation medium values from the range were preferred.
- The dropout also dint have much importance similar to the number of layers since bidirectional thing seems to be very much helping the network to learn better.
- Batch size smaller was preferred but the importance of that to the accuracy was low. So a smaller batch size 32 was added

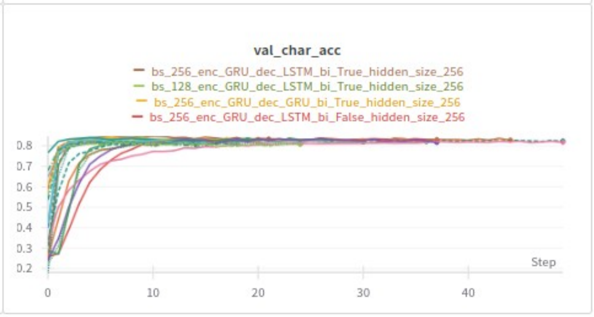
Keeping all this in mind the next sweep with the bayesian method of sweep is configured.

```
sweep_config = {
    "name" : "Vanilla Seq2Seq bayes",
    "method" : "bayes",
    "metric" : {
        "name" : "val_char_acc",
        "goal" : "maximize"
    },
    "parameters" : {
        "learning_rate" : {"values" : [0.001]},
        "dropout_rnn" : {"values" : [0, 0.1, 0.2, 0.4]},
        "batch_size" : {"values" :  [32, 64, 128, 256]},
        "epochs" : {"values" : [40]},
        "embedding_dim" : {"values" : [16, 32, 64, 128, 256]},
        "num_layers" : {"values" : [1, 2, 3, 5]},
        "hidden_size_enc" : {"values" : [32, 64, 128, 256]},
        "enc_cell_type" : {"values" : ["LSTM", "GRU"]},
        "dec_cell_type" : {"values" : ["LSTM", "GRU"]},
        "bi_directional" : {"values" : [True, False]}
    }
}
```



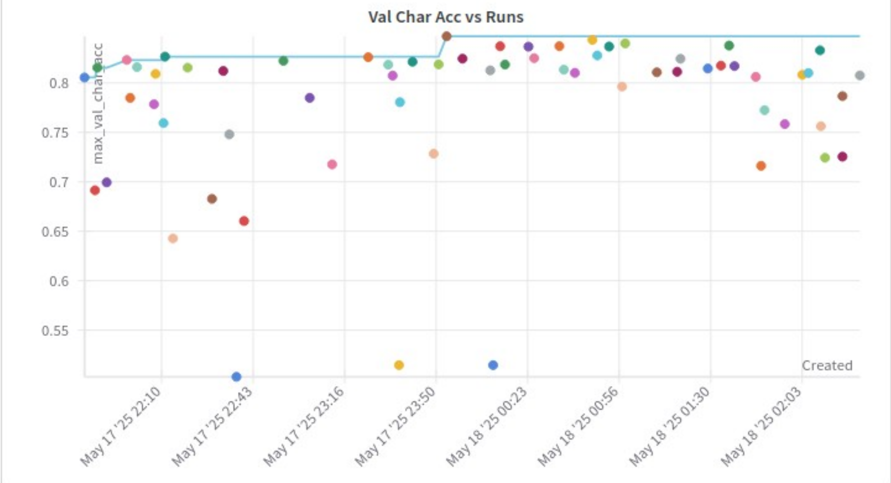


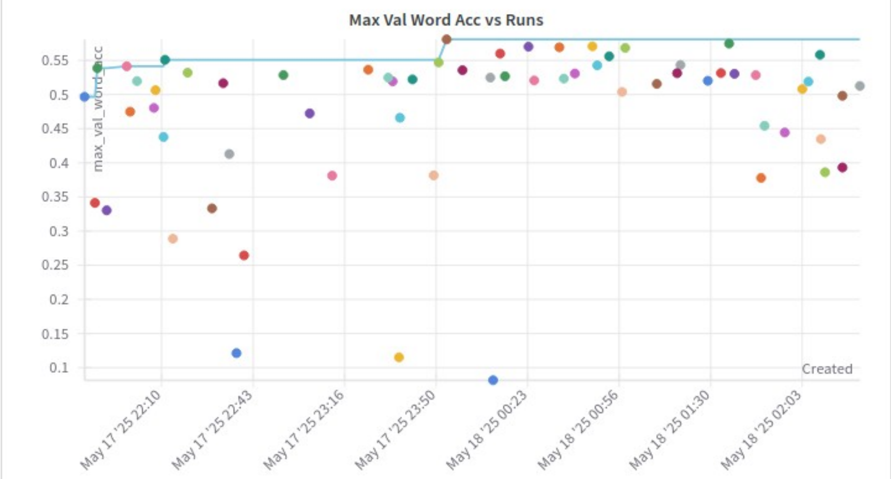


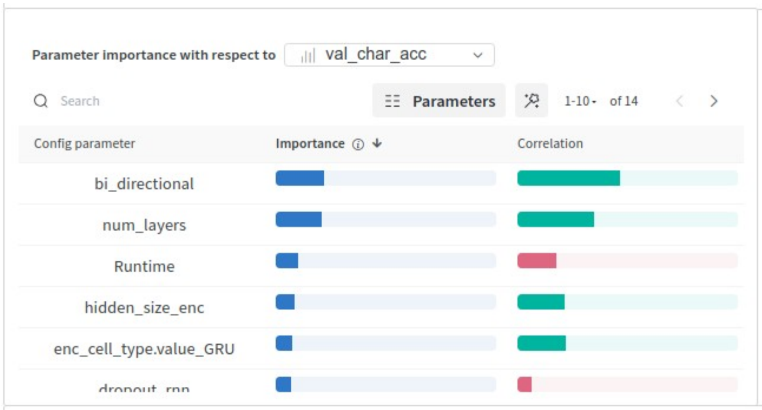


✓ Sweep: Vanilla Seq2Seq bayes 1 65

☐ Sweep: Vanilla Seq2Seq bayes 2 0



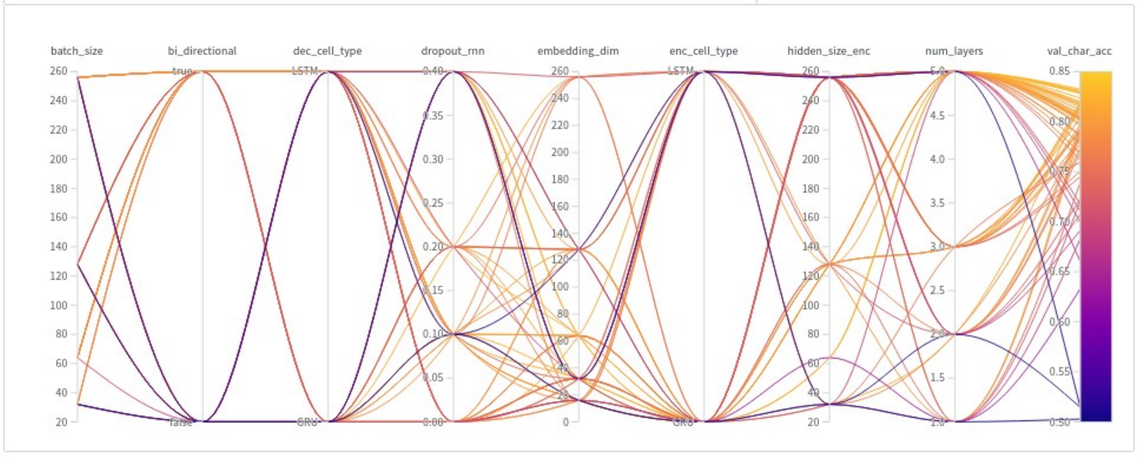


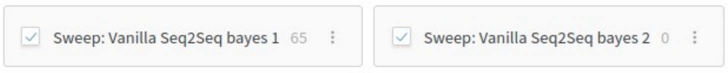


Parameter importance with respect to val_char_acc

Search | Parameters | 1-10 of 14

Config parameter	Importance	Correlation
bi_directional		
num_layers		
Runtime		
hidden_size_enc		
enc_cell_type.value_GRU		
dropout_rnn		





Sweep: Vanilla Seq2Seq bayes 1 65

Sweep: Vanilla Seq2Seq bayes 2 0

Best Model Parameters and Accuracy :

- Batch size : 256, Bi_directional : True, Decoder_cell : LSTM, Encoder_cell : GRU, Dropout : 0.4, Embedding dim : 64, Hidden_size_state : 256, Number_layers : 5 (Train loss : 0.04483, Val loss : 0.42543, Val_char_acc : 0.8470, Val_word_acc : 0.5807)

Next Best 9 Runs:

- Batch size : 256, Bi_directional : True, Decoder_cell : GRU, Encoder_cell : GRU, Dropout : 0.4, Embedding dim : 128, Hidden_size_state : 256, Number_layers : 5 (Train loss : 0.0580, Val loss : 0.4210, Val_char_acc : 0.8434, Val_word_acc : 0.5706)
- Batch size : 128, Bi_directional : True, Decoder_cell : LSTM, Encoder_cell : GRU, Dropout : 0.4, Embedding dim : 32, Hidden_size_state : 256, Number_layers : 5 (Train loss : 0.05077, Val loss : 0.4458, Val_char_acc : 0.8399, Val_word_acc : 0.5683)
- Batch size : 128, Bi_directional : False, Decoder_cell : GRU, Encoder_cell : GRU, Dropout : 0.2, Embedding dim : 256, Hidden_size_state : 256, Number_layers : 5 (Train loss : 0.0846, Val loss : 0.44244, Val_char_acc : 0.8377, Val_word_acc : 0.5746)
- Batch size : 32, Bi_directional : False, Decoder_cell : GRU, Encoder_cell : GRU, Dropout : 0.1, Embedding dim : 32, Hidden_size_state : 64, Number_layers : 5 (Train loss : 0.15683, Val loss : 0.37857, Val_char_acc : 0.8371, Val_word_acc : 0.5692)
- Batch size : 256, Bi_directional : False, Decoder_cell : LSTM, Encoder_cell : GRU, Dropout : 0.2, Embedding dim : 64, Hidden_size_state : 256, Number_layers : 5 (Train loss : 0.0691, Val loss : 0.4242, Val_char_acc : 0.8370, Val_word_acc : 0.5599)
- Batch size : 128, Bi_directional : True, Decoder_cell : GRU, Encoder_cell : GRU, Dropout : 0.4, Embedding dim : 128, Hidden_size_state : 256, Number_layers : 5 (Train loss : 0.0907, Val loss : 0.4264, Val_char_acc : 0.8367, Val_word_acc : 0.5560)
- Batch size : 128, Bi_directional : False, Decoder_cell : LSTM, Encoder_cell : GRU, Dropout : 0.2, Embedding dim : 128, Hidden_size_state : 256, Number_layers : 5 (Train loss : 0.06545, Val loss : 0.4821, Val_char_acc : 0.8366, Val_word_acc : 0.5700)

- Batch size : 256, Bi_directional : True, Decoder_cell : LSTM, Encoder_cell : GRU, Dropout : 0.1, Embedding dim : 16, Hidden_size_state : 256, Number_layers : 3 (Train loss : 0.0292, Val loss : 0.4340, Val_char_acc : 0.8328, Val_word_acc : 0.5582)
- Batch size : 64, Bi_directional : True, Decoder_cell : LSTM, Encoder_cell : GRU, Dropout : 0.1, Embedding dim : 16, Hidden_size_state : 128, Number_layers : 5 (Train loss : 0.04877, Val loss : 0.4455, Val_char_acc : 0.8277, Val_word_acc : 0.5426)

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.

SOL :

From the previous questions plot the inferences are made and noted...

From the first sweep the following were noted (copied from the previous question)

Inferences

- From the correlation plot we could directly see that for non attention encoder-decoder models bidirectional nature helps a lot.
- SGD optimizer was bad and was removed in the next following sweep. Most of the runs with SGD dint train well. On the other hand Adam optimizer was good and started to learn quickly.
- Encoder RNN was bad when compared to LSTM and GRU, decoder having RNN were somewhat okay but not the best. Having RNN takes slightly more epochs to start training and converge. The top 5 best models of this sweep did not have RNN as either of the decoder or encoder. Thus RNN was removed.
- Having hidden size (hidden state dimension) slightly larger is preferred when compared to the smaller dimension.
- The embedding dimension for input layer is not of very much importance from the covariance plot but from the correlation medium values from the range were preferred.
- The dropout also dint have much importance similar to the number of layers since bidirectional thing seems to be very much helping the network to learn better.
- Batch size smaller was preferred but the importance of that to the accuracy was low. So a smaller batch size 32 was added

From the bayesian sweep and the top 10 runs attached in the previous question the final inferences are written below,
Inferences

- From the covariance plot and the top 10 runs we can see that more number of layers in the encoder and decoder layers the better was the final trained accuracy. Also note that 9 out of the 10 best runs had 5 layers.
- Setting Bidirectional as true helps a lot in the non-attention case increasing the accuracy of the trained model. The correctional plot suggest that Bidirectional has a lot of importance and positive correlation suggesting to set it to true.
- Larger size for the hidden state is preferred as can be seen from the covariance plot also from the top 10 runs we could see that 8 of them are having hidden state size to be 512.
- Considering dropout medium dropout was preferred from the correlation plot and the top 10 runs while the top few ones has 0.4 dropout all of the lower ones has dropout lesser. Thus dropout helps here since there the five number of layers in the encoder and decoder.

- Larger Batch size were preferred as in the top 10 runs many of the batch size were around 128 and some being 256. Thus from the correlation plot as well medium range of batch size is preferred which was 128.
- Considering the embedding dimension for the input were not of much importance as can be seen the values varies as we compare among the top 10 best runs from the bayesian sweep.
- Considering the cell type both LSTM and GRU both of them we good, but from the top 10 runs we could see that all of them has GRU as the encoder. If given a task to have same cell type for both encoder and decoder for easier build GRU-GRU is the best for non-attention Tamil-English transliteration task (from the evidences presented above).

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!
- ...

SOL :

(a) Best Model Vanilla Seq2Seq

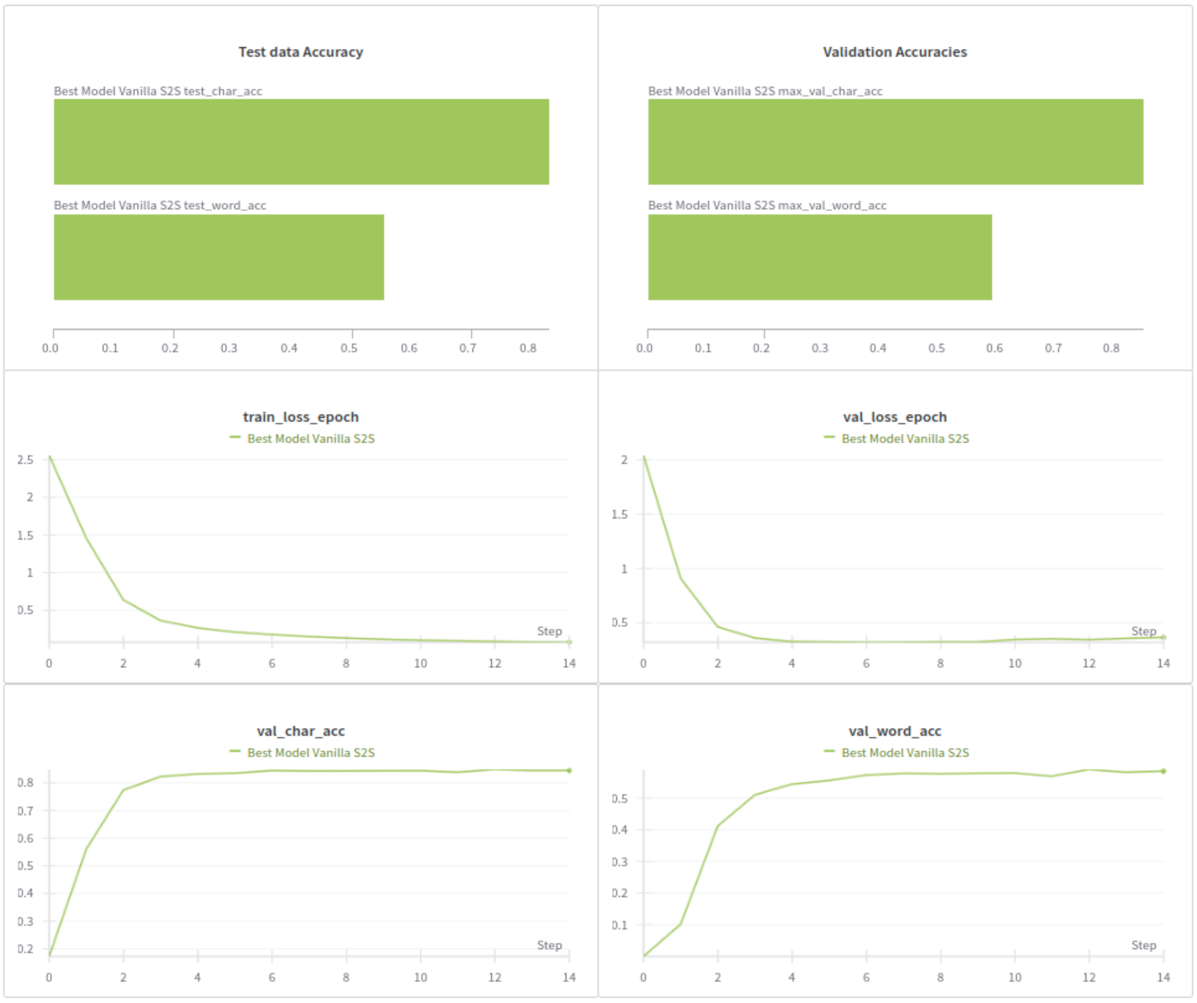
The best model obtained from the previous question is taken and retrained and the logged results are attached as follows,

The configuration:

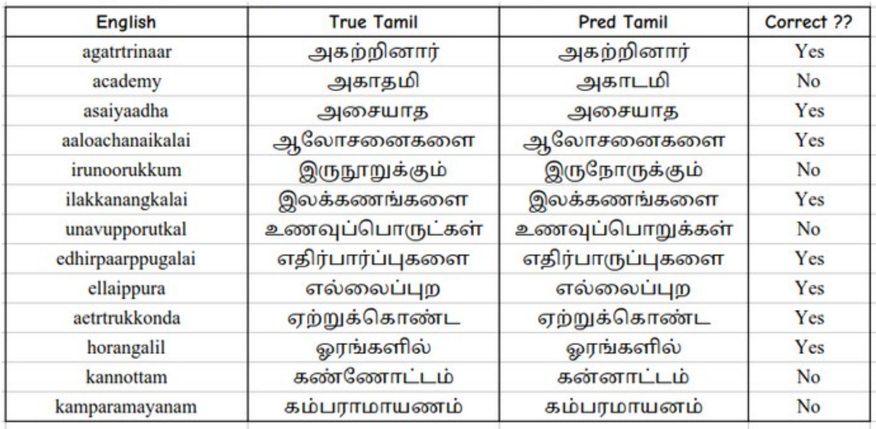
```
config = {
        "learning_rate" : 0.001,
        "dropout_rnn" : 0.4,
        "batch_size" :  256,
        "epochs" : 15,
        "embedding_dim" : 64,
        "num_layers" : 5,
        "hidden_size_enc" : 256,
        "enc_cell_type" : "GRU",
        "dec_cell_type" : "LSTM",
        "bi_directional" : True,
    }
```

The Accuracy of the model is given below,

- Word-wise accuracy (test) : 0.5531
- Character-wise accuracy (test) : 0.8291
- Word-wise accuracy (val) : 0.5901
- Character-wise accuracy (val) : 0.8480



(b) Sample Inputs and predictions Some sample inputs and their predictions are attached. While going through the test data predictions I was able to see some things which have ground truth wrong and some mis-interpretations which we will see in the next section. Some of the sample input and its predictions are attached. The predictions as a csv file is saved in GitHub project.



English	True Tamil	Pred Tamil	Correct ??
agatrtrinaar	அகற்றினார்	அகற்றினார்	Yes
academy	அகாதமி	அகாடமி	No
asaiyaadha	அசையாத	அசையாத	Yes
aaloachanaikalai	ஆலோசனைகளை	ஆலோசனைகளை	Yes
irunoorukkum	இருநூறுக்கும்	இருநோருக்கும்	No
ilakkanangkalai	இலக்கணங்களை	இலக்கணங்களை	Yes
unavupporutkal	உணவுப்பொருட்கள்	உணவுப்பொறுக்கள்	No
edhirpaarppugalai	எதிர்பார்ப்புகளை	எதிர்பாருப்புகளை	Yes
ellaippura	எல்லைப்புற	எல்லைப்புற	Yes
aetrtrukkonda	ஏற்றுக்கொண்ட	ஏற்றுக்கொண்ட	Yes
horangalil	ஓரங்களில்	ஓரங்களில்	Yes
kannottam	கண்ணோட்டம்	கன்னாட்டம்	No
kamparamayanam	கம்பராமாயணம்	கம்பரமாயனம்	No

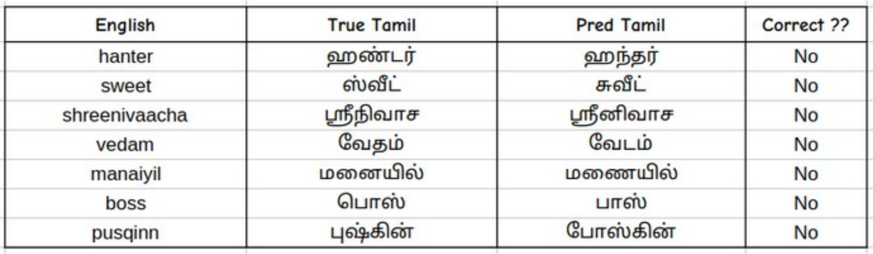
Figure - 01 : Sample Input Output pairs and predictions made by the model

(c) Errors in the prediction test and reasoning

Since we are training the model for transliteration task the English characters are transliterated but a sequence of English characters may represent more than one letter in native language

Some examples

- "ra" is the same sound made by two Tamil letters "ர" and "ற"
- "na" is the same sound made by three Tamil letters "ந", "ண" and "ன"
- Even though "ta" is the corresponding to "ட" and "tha" corresponding to "த" i see "ta" is used for both the tamil characters mentioned.
- "la" is the same sound for "ல" and "ள" i see this is not confused much as the position where it is occurring is different the former is mostly appearning in middle of word where as latter is mostly towards the end of the word.
- Some were early stopped as well.
- Some of the longer/shorter sound were not correctly predicted. The examples for the above mentioned mistakes is attached below.



English	True Tamil	Pred Tamil	Correct ??
hanter	ஹண்டர்	ஹந்தர்	No
sweet	ஸ்வீட்	சுவீட்	No
shreenivaacha	ஸ்ரீநிவாச	ஸ்ரீனிவாச	No
vedam	வேதம்	வேடம்	No
manaiyil	மனையில்	மணையில்	No
boss	பொஸ்	பாஸ்	No
pusqinn	புஷ்கின்	போஸ்கின்	No

Figure - 02 : Sample Input-Output prediction pair justifying the reasoning behind prediction

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).

SOL :

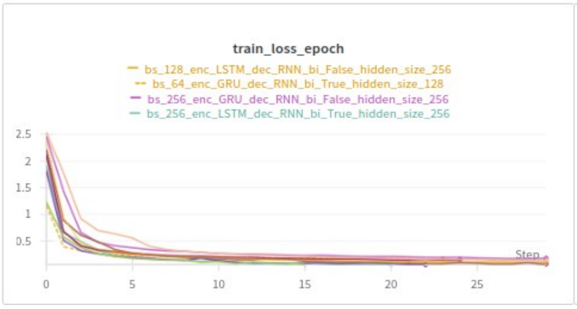
(a) Yes, The plots and the configurations are attached below,

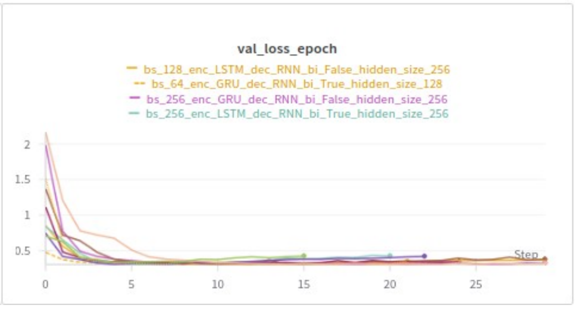
Since the attention based encoder decoder models have more parameters and computations I have directly went forward with the bayes sweep with the parameter configurations given below, Also i have done a random sweep layer with better inferences on wjar not to do. First lets look at a better one.

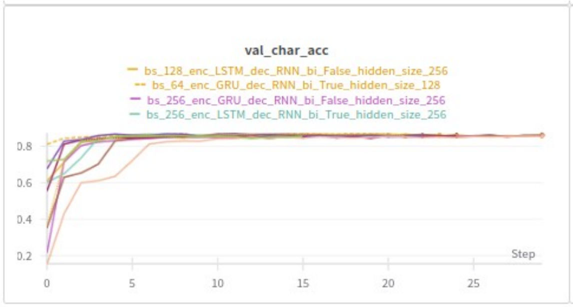
Note : There are two metrics that i have used word-wise accuracy and character-wise accuracy the sweep uses character-wise accuracy to maximize but i have logged the word-wise accuracy as well which will be seen in plots. Also the max epochs were 30 and incorporated early-stopping with a patience parameter of 7 so which prunes runs which dint improve over 7 epochs. Also note that teacher forcing method was used in training decoder and greedy search was used while calculation accuracy. The accuracy and the name of the metrics and their definitions are same as the vanilla Seq2Seq model

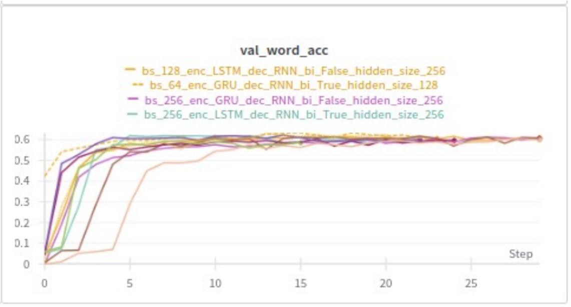
```
sweep_config = {
    "name" : "Attention Seq2Seq bayes",
    "method" : "bayes",
    "metric" : {
        "name" : "val_char_acc",
        "goal" : "maximize"
    },
    "parameters" : {
        "learning_rate" : {"values" : [0.001]},
        "dropout_rnn" : {"values" : [0, 0.1, 0.2, 0.4]},
        "batch_size" : {"values" :  [32, 64, 128, 256]},
        "epochs" : {"values" : [30]},
        "embedding_dim" : {"values" : [32, 64, 128, 256]},
        "num_layers" : {"values" : [1, 2, 3, 5]},
        "hidden_size_enc" : {"values" : [64, 128, 256]},
        "enc_cell_type" : {"values" : ["RNN", "LSTM", "GRU"]},
        "dec_cell_type" : {"values" : ["RNN", "LSTM", "GRU"]},
        "bi_directional" : {"values" : [True, False]}
    }
}
```

The plots for the bayesian sweep is attached as follows,



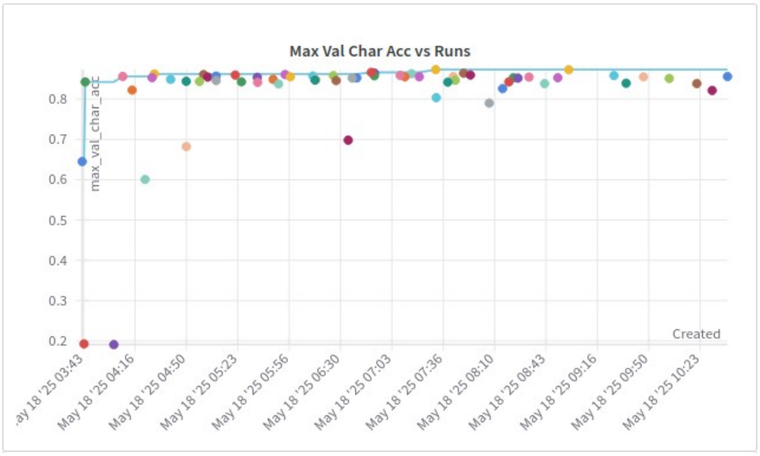


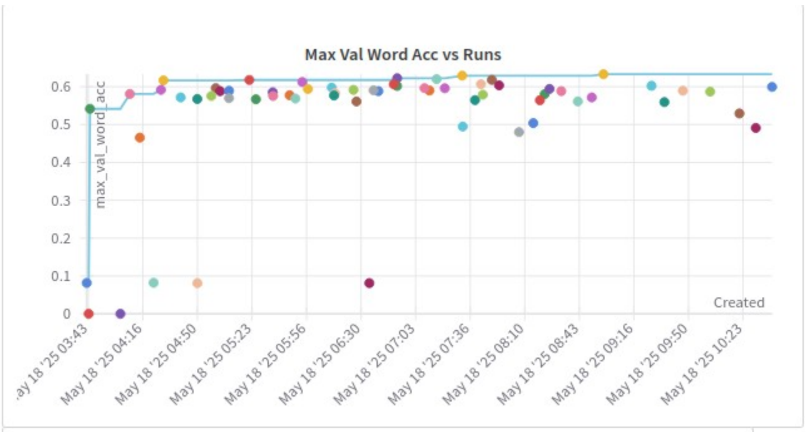


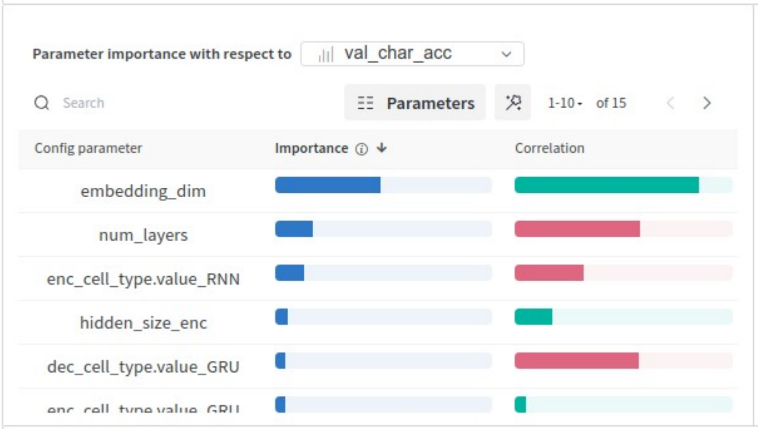


✓ Sweep: Attention Seq2Seq bayes 1 66

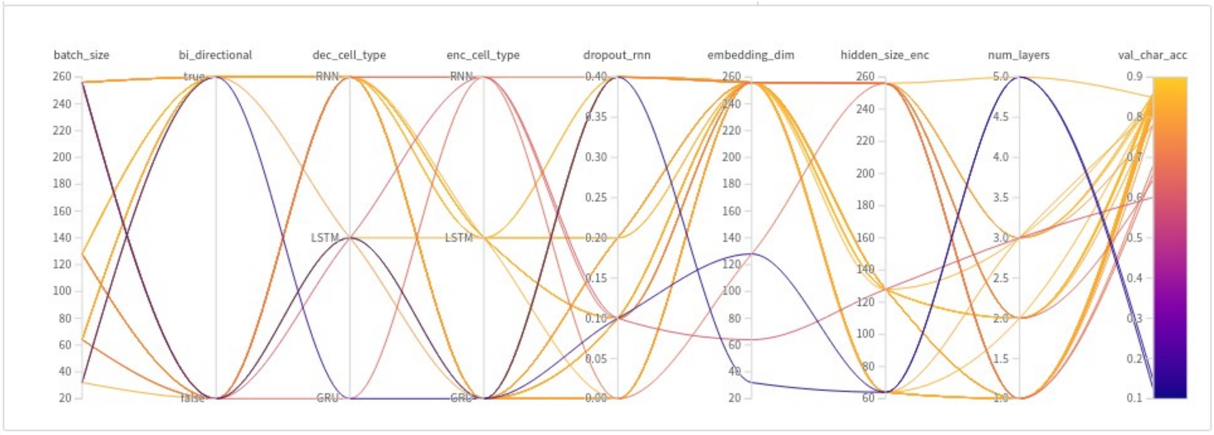
Sweep: Attention Seq2Seq bayes 2 0

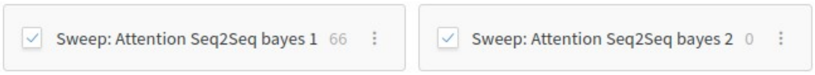






Config parameter	Importance	Correlation
embedding_dim		
num_layers		
enc_cell_type.value_RNN		
hidden_size_enc		
dec_cell_type.value_GRU		
enc_cell_type.value_GRU		




Sweep: Attention Seq2Seq bayes 1 66

Sweep: Attention Seq2Seq bayes 2 0

Some interesting inferences from these plots are as follows,

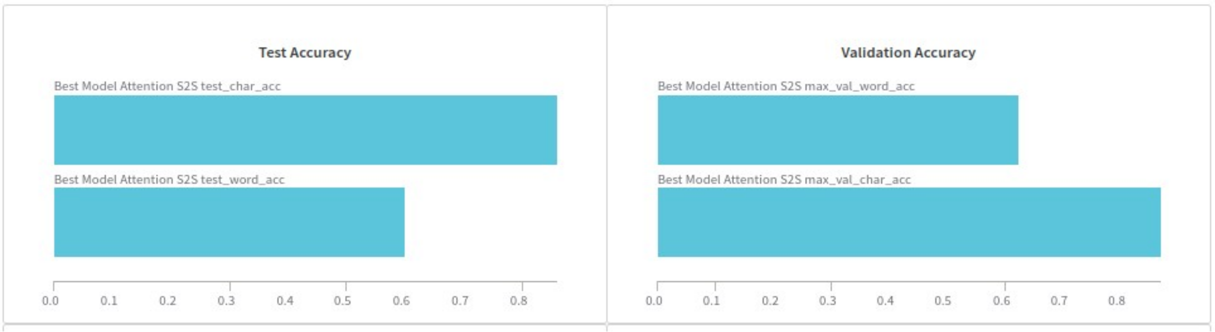
- Larger size embedding dimension was preferred from the correlation plots and the importance was high as well, might be with attention mechanisms better embedding was learnt.
- Compared to the vanilla Seq2Seq models here RNN is training well as we can see many RNN being used as decoder in the best configurations.
- Lower layers in the encoder and decoder were preferred might be due to the fact that attention helps better encoding of the states to be passed in the decoder. Might be the fact that choosing RNN as decoder and larger number of layers effectively dint train well.
- In the encoding step the RNN were not preferred and in the decoder step they work fine with the training and the accuracy.

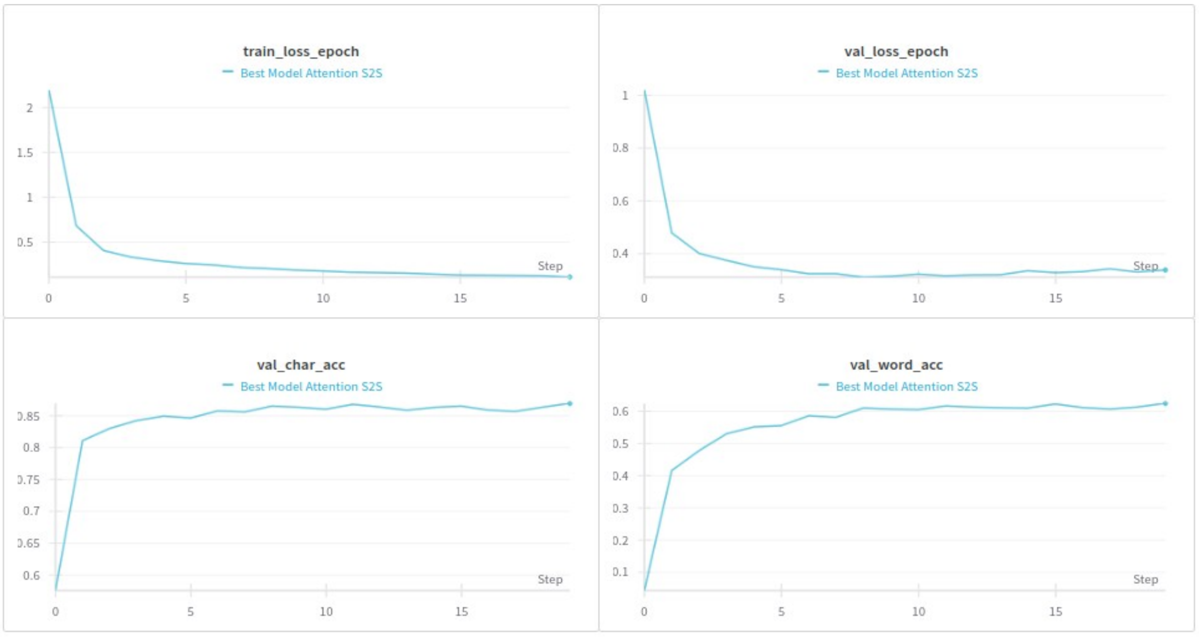
- Encoding hidden state dimension larger the better as suggested from the correlation plot and bi-directionality has little to no-effect because the possible reason is that attention takes care of the better encoding done when using bi-directional encoders.
- Smaller dropouts were preferred by the top performing models. Lets look at the best configurations.

Best Model Configurations :

- Batch_size:256, Bi_directional:true, Dec_cell_type:"RNN", Dropout_rnn:0.2, Embedding_dim:256, Enc_cell_type:"GRU", Epochs:30, Hidden_size_enc:128, Learning_rate:0.001, Num_layers:2

(b) The best model is retrained and attached below...





Run set 1

The model accuracy is given below,

- Word-wise accuracy (test) : 0.5996
- Character-wise accuracy (test) : 0.8589
- Word-wise accuracy (val) : 0.6244
- Character-wise accuracy (val) : 0.8693

Yes the attention based Encoder-Decoder model is better than the vanilla Seq2Seq models as the accuracy is 3%-4% percentage more in all categories. The predictions are in the folder predictions_attention as a csv file.

(c) Yes the attention based model is better than the vanilla model (This can also be seen from the sweep that almost all the configurations of the encoder-decoder models get trained well) and some the corrected results is attached below,

English	True Tamil	Pred Tamil Vanilla	Pred Tamil Attention	Attention Better
agaththikkeerai	அகத்திக்கீரை	அகத்திக்கீறை	அகத்திக்கீரை	Yes
agazhaaivil	அகழாய்வில்	அகழாயில்	அகழாய்வில்	Yes
kodhikkum	கொதிக்கும்	கோதிக்கும்	கொதிக்கும்	Yes
koatpaatukalin	கோட்பாடுகளின்	கோட்பாட்டுகளின்	கோட்பாடுகளின்	Yes
samprathaaya	சம்பிரதாய	சாம்பாதயா	சம்பிரதாய	Yes
dhandanaigal	தண்டனைகள்	தணணணைகள்	தண்டனைகள்	Yes
thamakkaena	தமக்கென	தமக்கென	தமக்கேன	No

Figure - 03 : Better predictions by the attention based encoder decoder models compared to Vanilla models

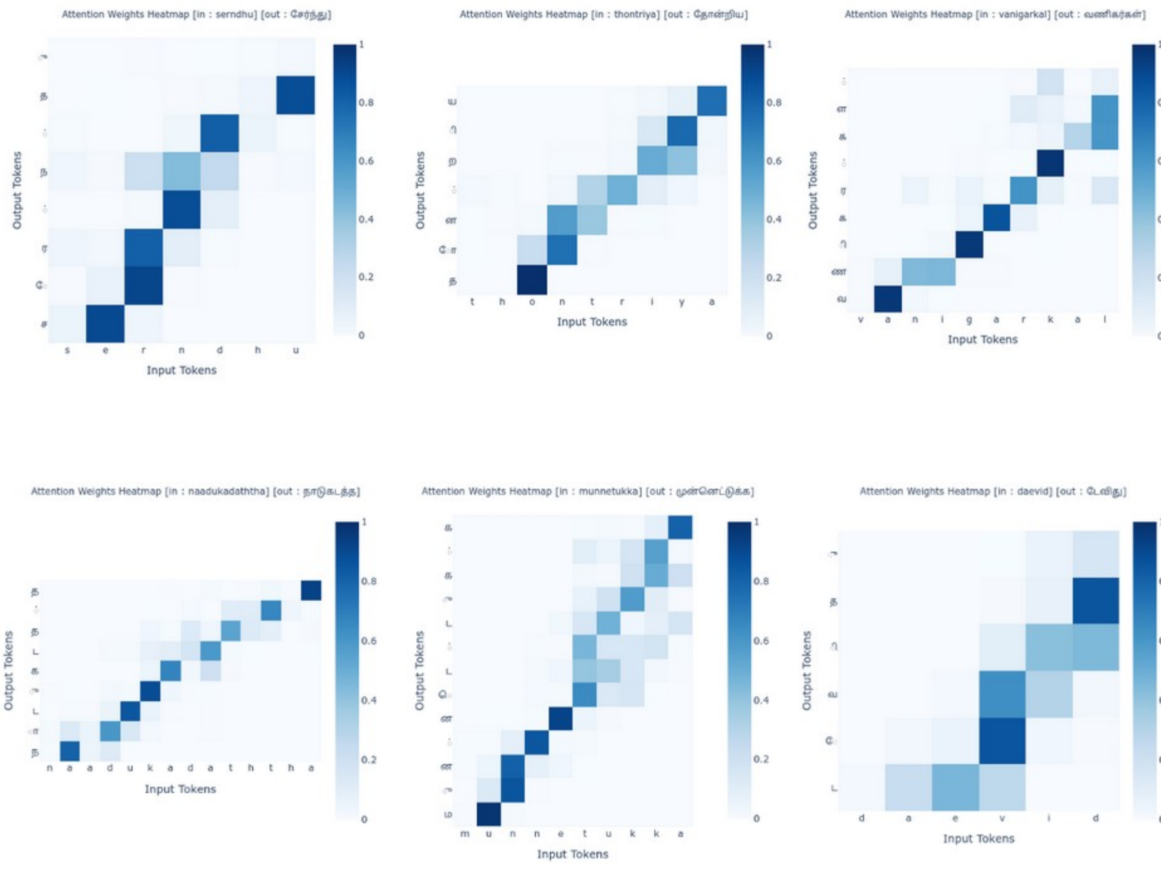
Inferences

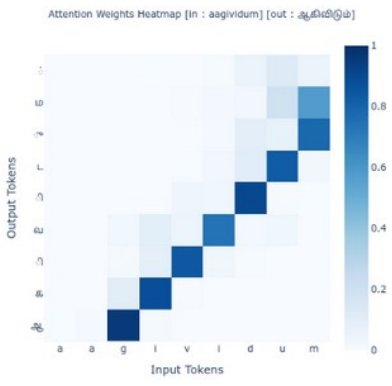
- From this we can see the problems mentioned in the previous questions are solved by the attention models to some extent that is same English letters for multiple Tamil characters as told attention based mechanism gives better representation of the encoded states that make it more aware of the surroundings and give the correct word prediction. Like as shown by the first, second, fourth examples
- Some other problem such as larger/shorter sound is also corrected as shown by the third example.
- Some predictions with messy vanilla predicts are also corrected by attention based models like the fifth one and 6th one.
- But note that they also perform worse in some cases when compared to vanilla method and one such example is attached like the last example.

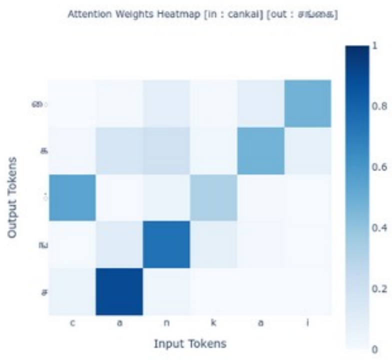
(d) Attention Heatmaps (For simplicity we will train a 1-layer network with same other parameters to represent the heatmaps).

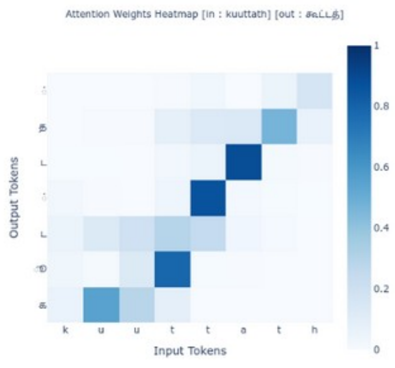
The attention weights for different inputs are added below, (These were done for 1 layer encoder decoder ...)

NOTE : These were done using the logic as following, the one layer encoder which is directional will give hidden states with respect to each input character, the attention weights are over these hidden state vectors for each decoder time step. So the characters in the x axis tells that this character's time step of the hidden state is given an attention weight by the decoder state character which is in the y-axis (They sum to 1 along rows !!!)









Question 6 (20 Marks)

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

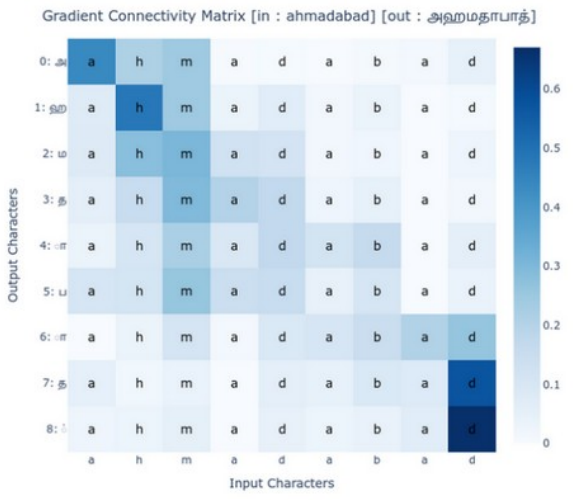
I like the visualisation in the figure captioned "Connectivity" in this article. Make a similar visualisation for your model. Please look at this blog for some starter code. The goal is to figure out the following: When the model is decoding the i-th character in the output which is the input character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

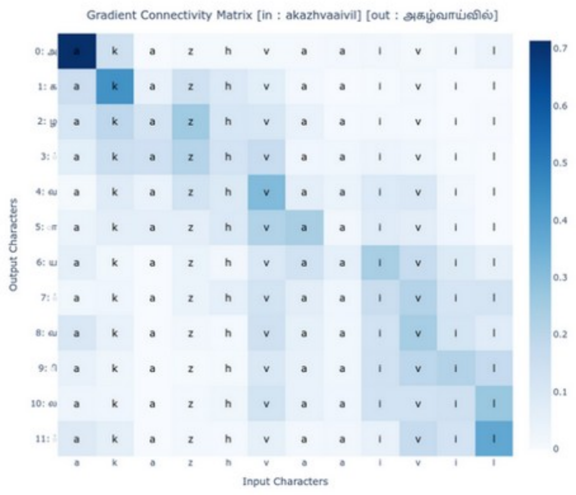
Have fun!

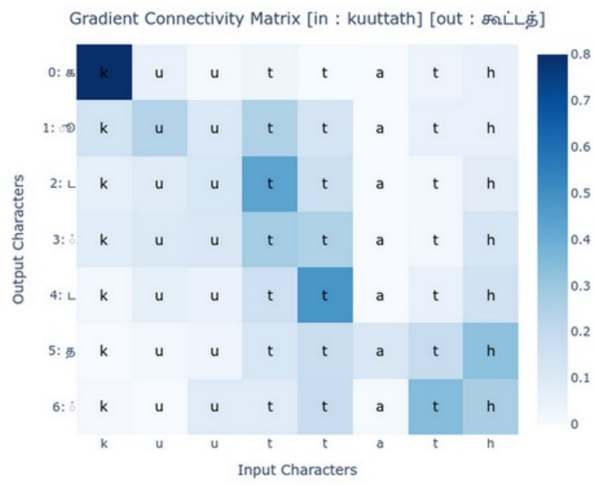
SOL :

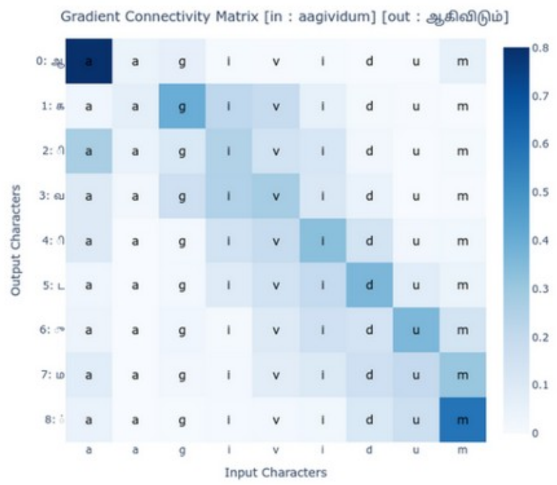
Now before attaching the plots let me tell how the connectivity weights were computed. For each input-output sequence, we compute a embedding of the input character which is 512 dimensional here, for each time step in decoder the output is passed the loss function (output.sum()) and the gradients with respect to the embedding dimension is found out. Now for each output character we will gave gradients with respect to each input character and 512 dimensional. Now we will take the norm across the 512 dimensional vector so each output character will have a grad norm with respect to each input character. Now this gradient is being scaled and normalized to be in range [0,1] and sum up to one. Then this is plotted as an heatmap to show the connectivity. The same model as the previous question is used 1 layer encoder-decoder with attention mechanism. Below are the plots attached.

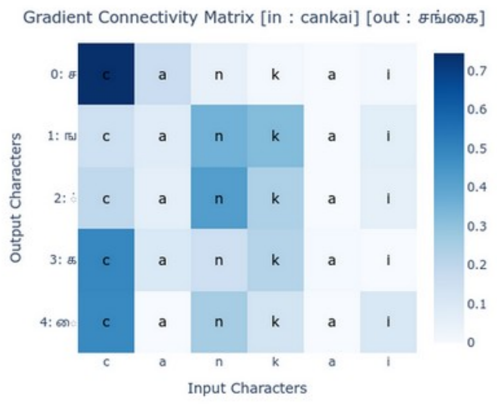
Note : Each row is the output character predicted by the greedy decoder search and their gradients are computed.

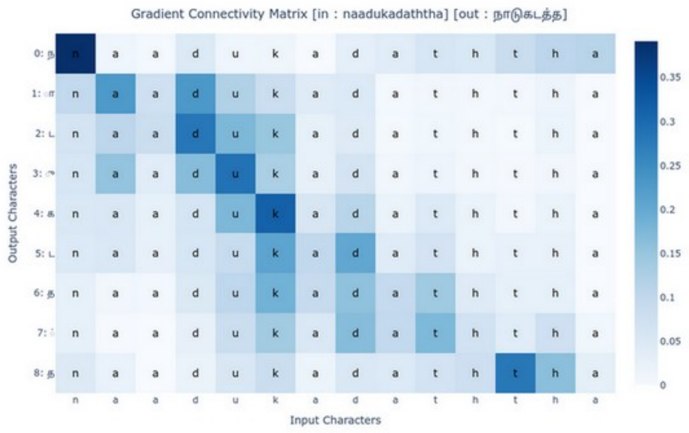


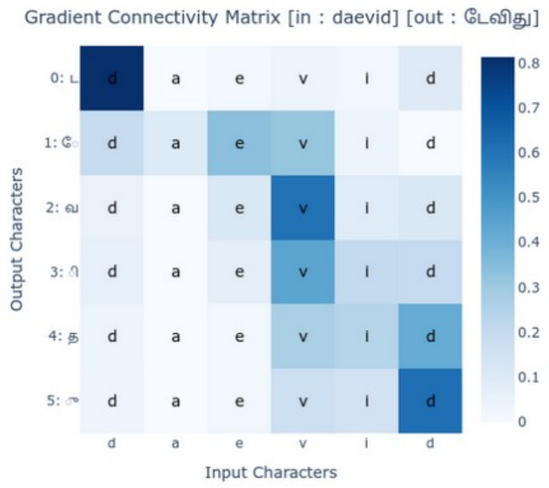


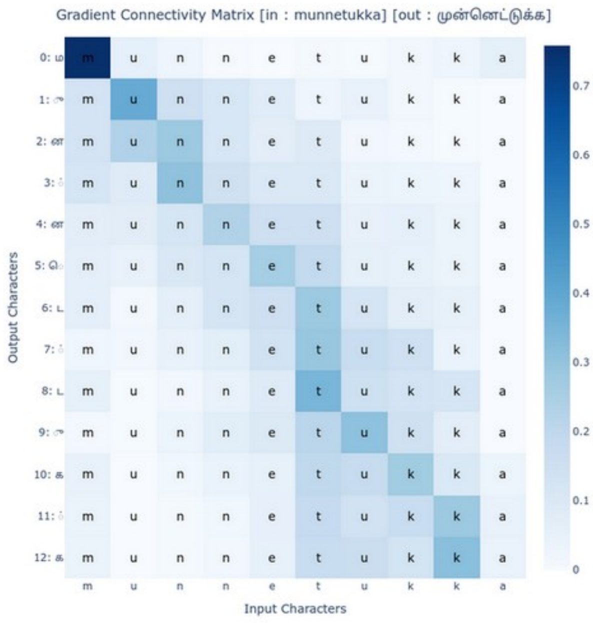

Gradient Connectivity Matrix [in : kuuttath] [out : கூட்டத்]
Output Characters
Input Characters

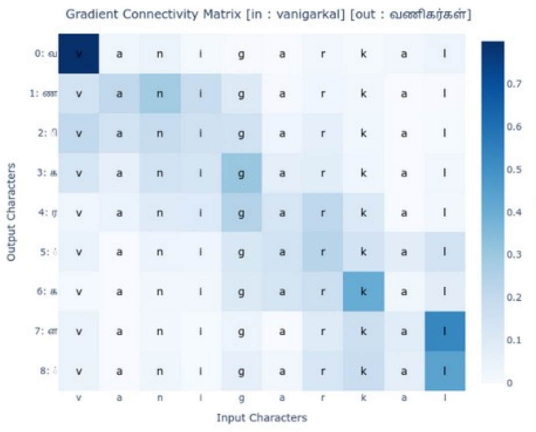

Gradient Connectivity Matrix [in : aagividum] [out : ஆகிவிடும்]
Output Characters
Input Characters

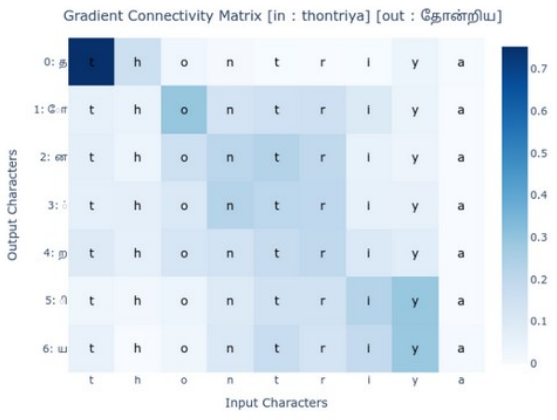

Gradient Connectivity Matrix [in : cankai] [out : சங்கை]
Output Characters
Input Characters


Gradient Connectivity Matrix [in : naadukadaththa] [out : நாடுகடத்த]
Output Characters
Input Characters


Gradient Connectivity Matrix [in : daevid] [out : டேவிது]
Output Characters
Input Characters


Gradient Connectivity Matrix [in : munnetukka] [out : முன்னெட்டுக்க]
Output Characters
Input Characters

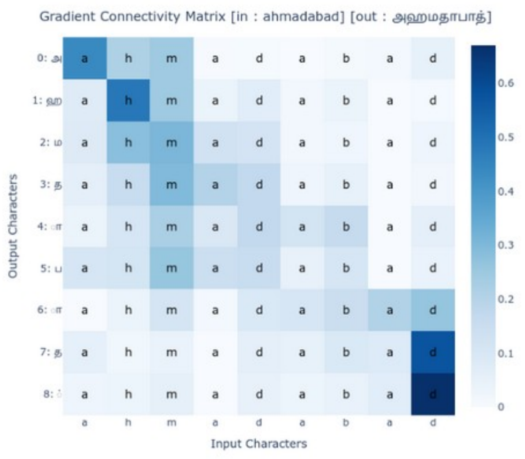


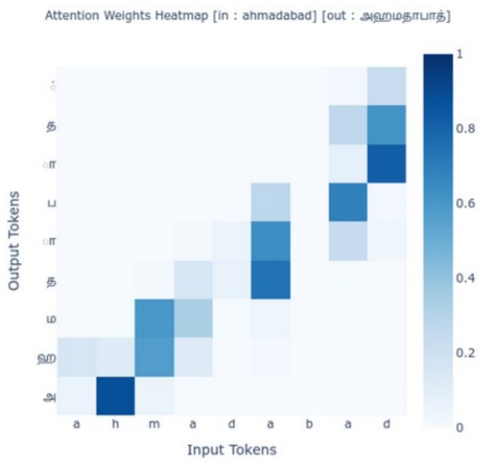


Even though these also look some what diagonal these have something interesting. Note that the difference between the attention heat map and this heat map will be shown with an image below. So going on with the interesting things,

- Tamil characters are matched with the equivalent English characters pretty well.
- I could see some long syllable is correctly given weights like in the last image right side, "த" is giving more attention to "t" and "h" and the next combined character "ே◌ா" is giving larger attention to the "o" .

Difference between Q5 and Q6





Even though these might look similar they are not the same,

- The first image is the connectivity measure computed by gradients mapping from each output character to input character
- The second image is the attention weight given to the encoding hidden state of the input time step to each of the output character.

Question 7 (10 Marks)

Paste a link to your github code for Part A

GitHub Link : https://github.com/AE21B105-JoelJ/DA6401_A03.git

Self Declaration

I, Joel J (Roll no: AE21B105), swear on my honor that I have written the code and the report by myself and have not copied it from the internet or other students.