

# CS6910 Assignment 3

## Report

cs22m062

Pankaj Singh Rawat cs22m062

Github link :

[https://github.com/pankajrawat9075/CS6910\\_assignment\\_3](https://github.com/pankajrawat9075/CS6910_assignment_3)

wandB report link : <https://api.wandb.ai/links/iitmadrass/d3qy9jox>

## Link to Report

## Question 1 (15 Marks)

Input embedding size	: m
Hidden cell state	: k
Length of i/p and output sequence	: T
Size of Vocabulary	: V

State computation equations for Recurrent Neural Networks are as below:

RNN :	$si = \sigma(Ux_i + Ws_{i-1} + b)$	$y_i =$
LSTM:	$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$ $i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$ $f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$	$s'_t =$ $s_t =$ $h_t =$
GRU:	$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$ $i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$	$s'_t =$ $s_t =$
Where, $U_{kxm}, W_{kxk}, b_{kx1}, U_{okxm}, W_{okxk}, b_{okx1}, U_{ikxm}, W_{ikxk}, b_{ikx1}, U_{fkxm}, W_{fkxk}, b_{fkx1}$		
For embedding layer, $e_i = Mv_i$ where, $M_{m \times V}$ is embedding matrix and		

(a) Computations (multiplications) done by the network: (Ignoring Embedding Layer)

RNN : Total Computations = Computations by encoder +  
Computations by decoder

$$= (km + k^2)T + (km + k^2 + km)T$$

LSTM : Total Computations = Computations by encoder +  
Computations by decoder

$$= (4(km + k^2) + 3k)T + (4(km + k^2) + 3k + mk)T$$

GRU: Total Computations = Computations by encoder +  
Computations by decoder

$$= (3(k^2 + km) + 3k)T + (3(k^2 + km) + 3k + mk)T$$

Computations for Embedding Layer (common for all 3) :  $2mV$

(b) Parameters of the network: (Ignoring Embedding Layer)

RNN : Total parameters = Parameters for encoder + Parameters for  
decoder

$$(km + k^2 + k) + (2km + k^2 + k + m)$$

LSTM : Total parameters = Parameters for encoder + Parameters for  
decoder

$$(4(k^2 + km + k)) + (4(k^2 + km + k) + km + m)$$

GRU : Total parameters = Parameters for encoder + Parameters for  
decoder

$$(3(k^2 + km + k)) + ((3(k^2 + km + k)) + km + m)$$

Parameters for Embedding Layer (common for all 3):  $2mV$

## Question 2 (10 Marks)

For this question we have used the Hindi language from the Aksharantar dataset.

The hyperparameter space on which sweeps are generated are as below :

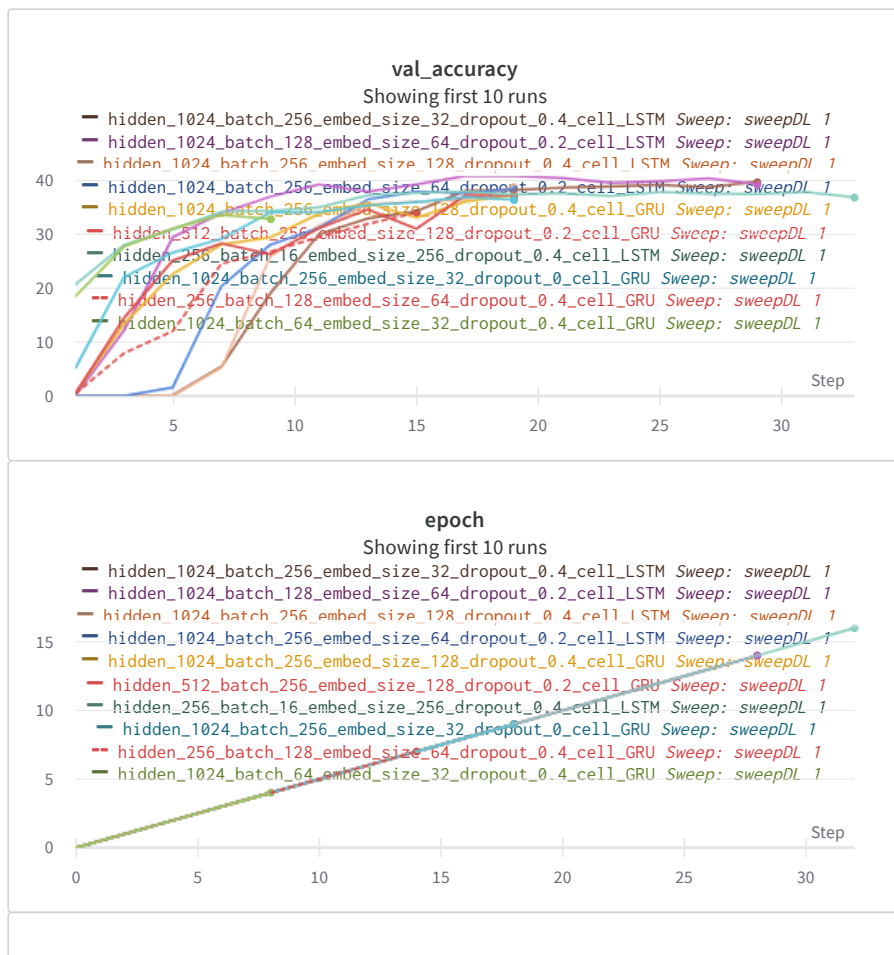
```
'learn_rate': {
  'values': [0.01, 0.001, 0.001]
},
'embedding_size': {
  'values': [32, 64, 128, 256, 512, 1024]
},
'batch_size':{
  'values':[16, 32, 64, 128, 256]
},
'hidden_size':{
```

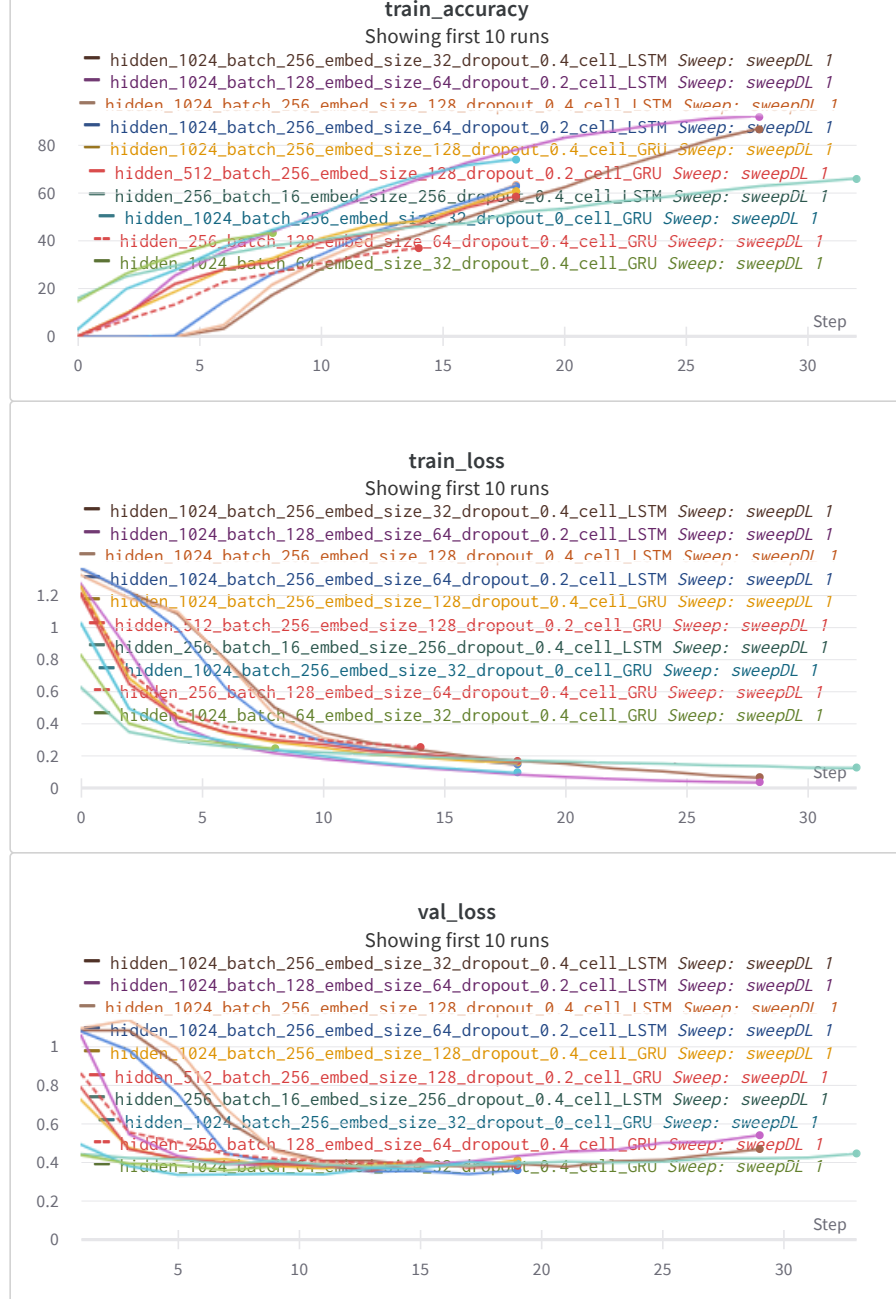
```

'values':[32, 64, 128, 256, 512, 1024]
},
'teach_ratio':{
  'values':[0.4, 0.5, 0.6]
},
'dropout':{
  'values':[0, 0.2, 0.4]
},
'cell_type':{
  'values':["RNN", "LSTM", "GRU"]
},
'bidirectional':{
  'values' : ["Yes","No"]
},
'num_layers_decoder':{
  'values': [1,2, 3, 4]
},
'num_layers_encoder':{
  'values': [1,2,3,4]
},
'epochs':{
  'values': [10, 15, 20, 25, 30]
},
'attention':{
  'values': ["Yes"]
}

```

## Plots for the sweeps





### Strategies used to obtain maximum performance from the runs:

0) Bayes Method : Finding the best hyperparameter is a tough task if sweep strategy is random, since there are exponential combinations of hyperparameters. Therefore I used the Bayes Method to sweep over the hyperparameters which saved a lot of time. I only ran sweep of 19 runs and was able to get best accuracy. Since I wasn't getting any improvement over multiple runs, I stopped at 19 only.

1)Optimizer Function: We didn't use all the values of each hyperparameter, for example, we read that the optimizer function that works best for the Recurrent Neural network was Adam, so rather than including all the optimizer functions we just have used Adam to reduce the number of runs.

2) Dropout: It is a regularization method for neural networks. Dropout was applied to the inputs to the RNN's and to LSTM memory cells (or GRU states), i.e. it drops out the input/update gate in LSTM/GRU. Drop out is added as an argument to layer so that it will mask the inputs and the Dropout layer after recurrent layer masks the outputs as well.

3)Data shuffling: Did shuffling of ordered input data, and found that data shuffling is a strong regularizer.

4) Early Stopping: A patience parameter of 5 runs was set, and once the validation accuracy is not seen to increase in the last 5 runs, the sweep is automatically stopped, with the guess that model is overfitting now and returns the last best-trained weights. But this did not give reasonable improvement on word-level accuracy as opposed to character-level accuracy.

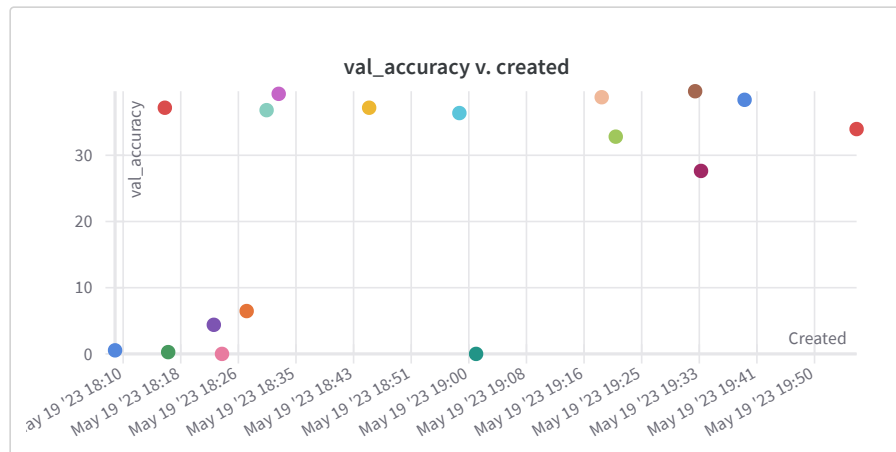
Code is made flexible enough such that the dimension of the input character embeddings, the hidden states of the encoders and decoders, the cell (RNN, LSTM, GRU), and the number of layers in the encoder and decoder can be changed.

## Best Hyperparameter Model

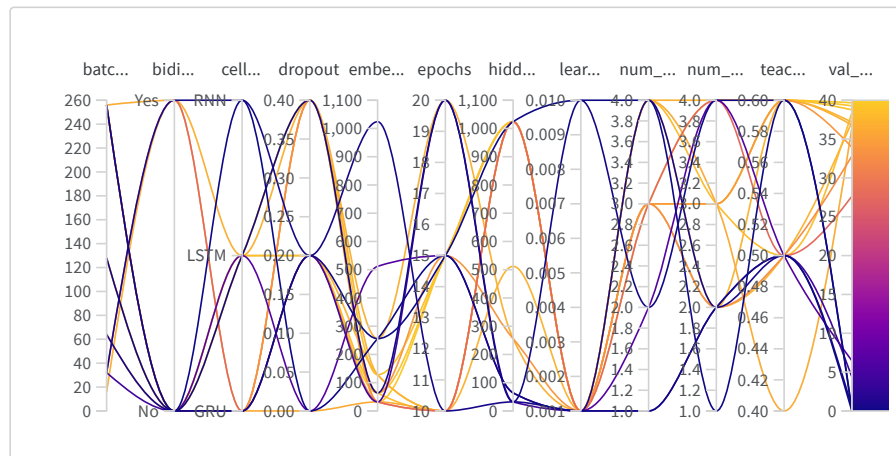
```
attention: No
batch_size: 256
bidirectional: No
cell_type: LSTM
dropout: 0.4
embedding_size: 32
epochs: 15
hidden_size: 1024
learn_rate: 0.001
num_layers_decoder: 3
num_layers_encoder: 3
teach_ratio: 0.6
```

Validation Accuracy for this Configuration was 39.3769 %.

## accuracy vs created plot



## parallel co-ordinates plot



## correlation summary table

Parameter importance with respect to <span>val_accuracy</span>		
Q Search	Parameters	1-10 of 17 < >
Config parameter	Importance  ↓	Correlation
hidden_size		
learn_rate		
num_layers_decoder		
embedding_size		
cell_type.value_RNN		
num_layers_encoder		
teach_ratio		
dropout		

## Question 3 (15 Marks)

Lesser the learn rate, better is the accuracy

- Evident from the high negative co-relation(-0.597 with validation accuracy) from co-relation summary table.
- Most models with higher val\_accuracy have learning rate of 0.001(evident from the parallel co-ordinate plot)

- When the learning rate is set too high, it can lead to the phenomenon known as "overshooting" or "diverging" during the training process. The model's parameters are being updated in large steps, causing the optimization process to become unstable and unable to converge to a good solution.

Using smaller sizes for the hidden layer does not give good results  
The more the hidden size better is the accuracy

- Hidden size parameter has the highest importance among all hyperparameters, with importance of 0.311 and positive co-relation of 0.368(data taken from co-relation summary table)
- The hidden size determines the number of parameters and the representational capacity of the model. By increasing the hidden size, we allow the model to learn more complex and nuanced patterns in the data. It seems to be the case for Hindi dataset that this increased capacity is beneficial for capturing fine-grained details and complex structure between the characters

Higher embedding size doesn't always lead to high accuracy

- There is a negative co-relation(-0.522) between accuracy and embedding size, seen in the co-relation summary table
- With importance of 0.079, it is evident that embedding size, doesn't influence validation accuracy much.
- In our task, it seems that learning fine-grained details between nearby characters is enough and the model need not learn fine-grained details between far away characters.
- Once a fine representation is learned for some embedding size, learning more complex representation of characters may prove to provide less accuracy than the former due to overfitting on train data.

LSTM and GRU performs way better than RNN

- In parallel co-ordinates plot, almost all RNN models give less validation accuracy(all lines passing through RNN cell type are bluish, less validation accuracy) while for LSTM/GRU cell type it's mostly yellowish(high validation accuracy)
- In co-relation summary table, RNN cell type has high negative co-relation to val\_accuracy(-0.400).
- All models with validation accuracy  $\geq 35\%$  are either LSTM or GRU(parallel co-ordinates plot).
- With the help of LSTMs and GRUs, we can deal with the problems of vanishing gradients of RNN by introducing some gates like input gates and forget gates, which allows better control over the gradient flow and also enables better preservation of "long-range dependencies". There exists one more problem with RNN which is a

long-range dependency that can be easily resolved by increasing the number of repeating layers in LSTM/GRU. Because of such disadvantages in RNN, LSTM/GRU is better than RNN.

More layers stacked in the network(decoder and encoder) improves val

- We can see the positive co-relation between the validation accuracy and number of layers in encoder(-.265) and decoder(0.390). from the co-relation summary table.
- Adding more layers increases the number of parameters in the model, allowing it to capture more complex patterns and representations in the input and output sequences
- Deeper architectures can help the model propagate information more effectively across longer sequences, allowing it to capture dependencies that span over multiple timesteps.

RNN based model takes longer time to converge than GRU or LSTM

- We suppose the lower learning accuracy of our RNN models evident from the **parallel plot** is due to this.
- Exploding gradients is a problem faced by RNN, where large error gradients accumulate and result in very large updates to neural network model weights during training. As a result, our model may remain unstable and unable to learn from the training data.
- Vanishing Gradients is another problem faced by RNN, where the gradient will be vanishingly small, effectively preventing the weight from changing its value. Because of these two problems, RNN takes quite more time to converge as compared to the other two.

Bidirectional doesn't really effect the validation accuracy

- Most of high accuracy models have bidirectional hyperparameters as either Yes or No (parallel plot).
- very less co-relation with val\_accuracy and we see hardly any importance of bidirectional value on accuracy(co-relation summary table)
- It seems that for this task, the relation of characters from right to left is not significant for accurate predictions.

dropout leads to better performance

- High positive co-relation of dropout with validation accuracy(0.308). Evident from co-relation summary table.
- dropout is a useful regularization technique that can improve the generalization ability of seq2seq models and help prevent overfitting. By introducing noise and encouraging independence among neurons, dropout promotes more robust learning and can lead to better performance on unseen data.



# Question 4(10 Marks)

a)

## Best Hyperparameter Model


```
attention: No
batch_size: 256
bidirectional: No
cell_type: LSTM
dropout: 0.4
embedding_size: 32
epochs: 15
hidden_size: 1024
learn_rate: 0.001
num_layers_decoder: 3
num_layers_encoder: 3
teach_ratio: 0.6
```

Validation Accuracy for this Configuration was 39.673 %.

test accuracy was 36.376953125 %

b)

- 1. Please find the "Prediction\_Venilla " folder in our github repo.
- 2. The following are the predictions:

runs.summary["data"] 

	input	predicted	Actual
1			
2			
3			
4			
5			
6			
7			
8			
9			

c)

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

- Most Of the Time it is getting confused between the Matras of Hindi . It is adding wrong Matras in hindi.

Example :- daanshil | दाँशिल | दानशील

Example :- jangadnaa | जंगदना | जनगणना

- I have observed that the model makes more error on vowels than on consonants. This is because of complexity of structure of vowels in Hindi. I too suffered from this when writing in Hindi.....

Example :- shurooh | शुरूह | शुरूः

Example :- takreeban | तकरीबान | तक्ररीबन

- It makes mistakes for characters that have almost same pronunciation

Example :- tikone | टिकोने | तिकोने

Example :- dibanu | दिबनन | डिबनू

- The model makes more errors on longer sequences.

Example :- hairatangen | हैराटेंजें | हैरतअंगेज़

Example :- upakhyaanon | उपख्यानों | उपाख्यानों

- The model makes more errors on longer sequences.

Example :- hairatangen | हैराटेंजें | हैरतअंगेज़

Example :- upakhyaanon | उपख्यानों | उपाख्यानों

- Vanilla Unable To work Properly For Matras.The consonants which have similar pronunciation as a vowel is predicted wrong by vanilla .

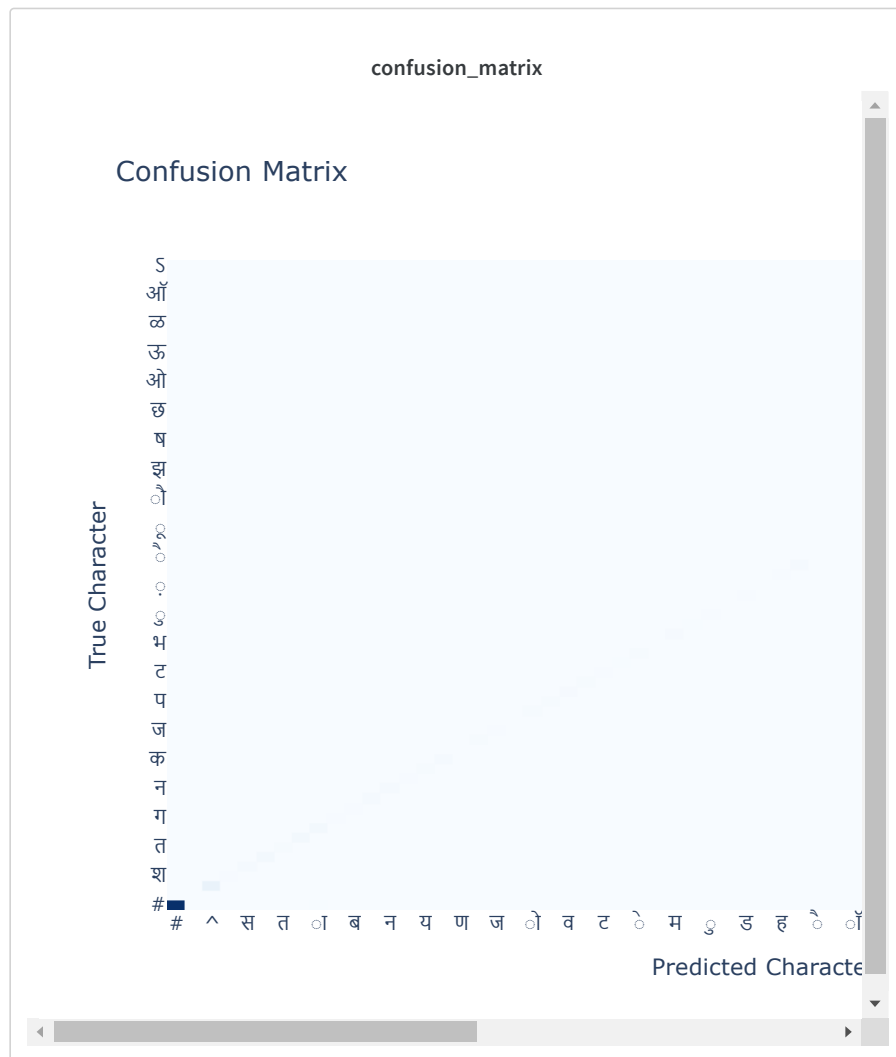
Example:- AMBAANI | अमबानी (Pred by Vennila)

- Vanilla is Unable to predict correctly for Vowel in the middle

Example:- ANGARAK | अंगररक(Pred by Vannila)

**Confusion matrix for the characters:**

Thing to remember : character to character prediction works well, more than 70% accuracy char to char accuracy was achieved.



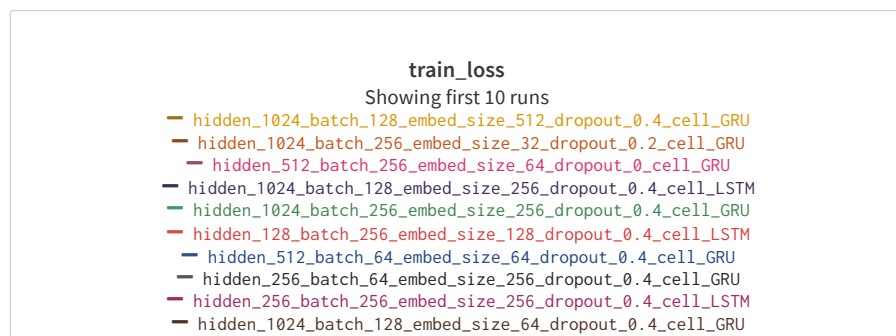
- It is evident from the confusion matrix that the char to char are predicted correctly, especially the normal characters(consonants).
- The matras are predicted with lower accuracy.

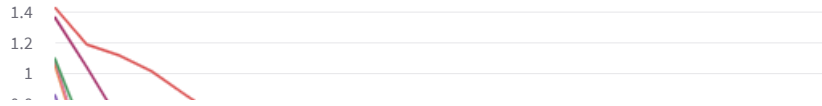
## Question 5(20 Marks)

a)

Yes I did tune the Hyperparameters again. Attention is a different architecture of learning than vanilla RNN, hence I did tune it.

Here are the Plots

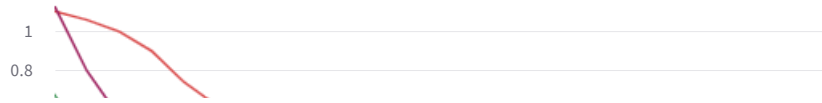




### val\_loss

Showing first 10 runs

- hidden\_1024\_batch\_128\_embed\_size\_512\_dropout\_0.4\_cell\_GRU
- hidden\_1024\_batch\_256\_embed\_size\_32\_dropout\_0.2\_cell\_GRU
- hidden\_512\_batch\_256\_embed\_size\_64\_dropout\_0\_cell\_GRU
- hidden\_1024\_batch\_128\_embed\_size\_256\_dropout\_0.4\_cell\_LSTM
- hidden\_1024\_batch\_256\_embed\_size\_256\_dropout\_0.4\_cell\_GRU
- hidden\_128\_batch\_256\_embed\_size\_128\_dropout\_0.4\_cell\_LSTM
- hidden\_512\_batch\_64\_embed\_size\_64\_dropout\_0.4\_cell\_GRU
- hidden\_256\_batch\_64\_embed\_size\_256\_dropout\_0.4\_cell\_GRU
- hidden\_256\_batch\_256\_embed\_size\_256\_dropout\_0.4\_cell\_LSTM
- hidden\_1024\_batch\_128\_embed\_size\_64\_dropout\_0.4\_cell\_GRU



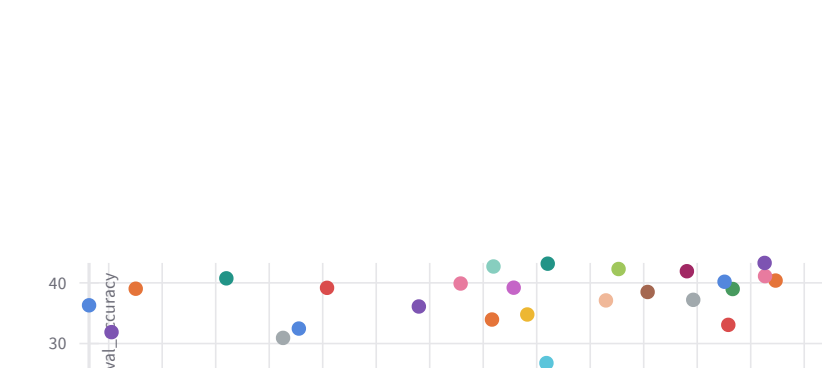
### val\_accuracy

Showing first 10 runs

- hidden\_1024\_batch\_128\_embed\_size\_512\_dropout\_0.4\_cell\_GRU
- hidden\_1024\_batch\_256\_embed\_size\_32\_dropout\_0.2\_cell\_GRU
- hidden\_512\_batch\_256\_embed\_size\_64\_dropout\_0\_cell\_GRU
- hidden\_1024\_batch\_128\_embed\_size\_256\_dropout\_0.4\_cell\_LSTM
- hidden\_1024\_batch\_256\_embed\_size\_256\_dropout\_0.4\_cell\_GRU
- hidden\_128\_batch\_256\_embed\_size\_128\_dropout\_0.4\_cell\_LSTM
- hidden\_512\_batch\_64\_embed\_size\_64\_dropout\_0.4\_cell\_GRU
- hidden\_256\_batch\_64\_embed\_size\_256\_dropout\_0.4\_cell\_GRU
- hidden\_256\_batch\_256\_embed\_size\_256\_dropout\_0.4\_cell\_LSTM
- hidden\_1024\_batch\_128\_embed\_size\_64\_dropout\_0.4\_cell\_GRU



### val\_accuracy v. created



### train\_accuracy

Showing first 10 runs

- hidden\_1024\_batch\_128\_embed\_size\_512\_dropout\_0.4\_cell\_GRU
- hidden\_1024\_batch\_256\_embed\_size\_32\_dropout\_0.2\_cell\_GRU
- hidden\_512\_batch\_256\_embed\_size\_64\_dropout\_0\_cell\_GRU
- hidden\_1024\_batch\_128\_embed\_size\_256\_dropout\_0.4\_cell\_LSTM
- hidden\_1024\_batch\_256\_embed\_size\_256\_dropout\_0.4\_cell\_GRU
- hidden\_128\_batch\_256\_embed\_size\_128\_dropout\_0.4\_cell\_LSTM
- hidden\_512\_batch\_64\_embed\_size\_64\_dropout\_0.4\_cell\_GRU
- hidden\_256\_batch\_64\_embed\_size\_256\_dropout\_0.4\_cell\_GRU
- hidden\_256\_batch\_256\_embed\_size\_256\_dropout\_0.4\_cell\_LSTM
- hidden\_1024\_batch\_128\_embed\_size\_64\_dropout\_0.4\_cell\_GRU



Parameter importance with respect to

val\_accuracy

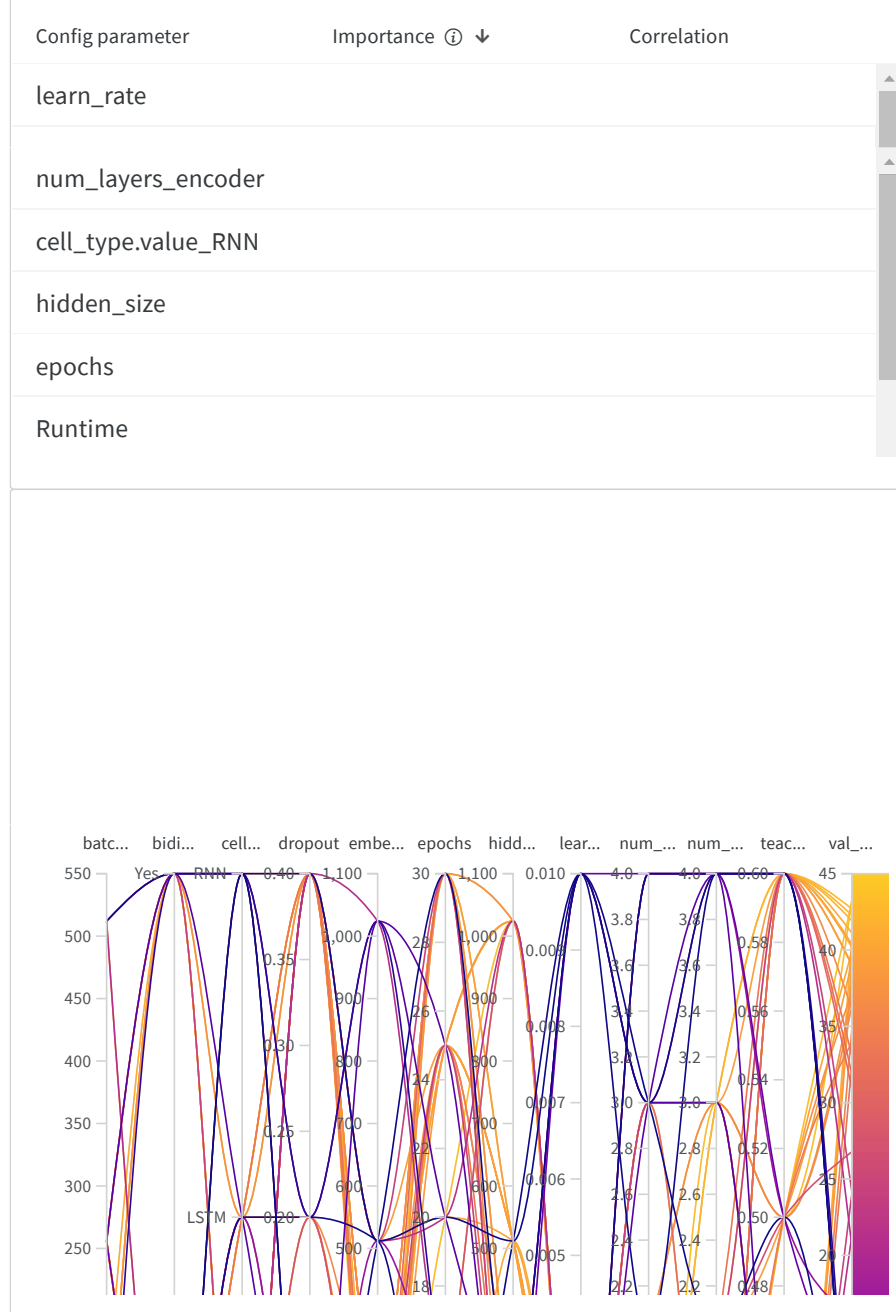
Q Search

Parameters



1-10 of 16





b)I have evaluated the best model and got accuracy as below

The best Hyperparameters were

```
'learn_rate' : 0.001,
'embedding_size': 32,
'batch_size': 64,
'hidden_size' : 1024,
'num_layers_encoder': 1,
'num_layers_decoder': 1,
'bidirectional': 'Yes',
'cell_type': "LSTM",
'teach_ratio': 0.5,
'dropout': 0.4,
```

```
'epochs': 20,  
'attention': "Yes"
```


test accuracy : 40.4296875 %

c)


Does the attention-based model perform better than the vanilla model?

Yes, attention performs better than vanilla model by about 4% higher accuracy on test data.

Predictions from the attention model

runs.summary["data"] 

	input	predicted	Actual
1			
2			
3			
4			
5			

 ← < 1 - 5 of 8192 > → Export as CSV Columns... Reset Table

some of the errors that attention model corrected and our inferences

(1)

- Attention mechanism is able to make correct predictions for long length words whereas vanilla model was struggling

Example :- input -> tirunelveli | correct output ->  
तिरुनेलवेली

vanilla prediction -> तिरुनेवेली | attention  
prediction-> तिरुनेलवेली

Example :- input -> dablyoopeedee | correct output ->  
डब्ल्यूपीडी

vanilla prediction -> दब्ल्यूपीडी | attention  
prediction-> डब्ल्यूपीडी

- Attention allows the model to consider all the characters of the word. This is especially helpful when the length of the word is large. Vanilla model only has one representation for the input that is passed to decoder, because of this information required to correctly predict long words is lost while passing for large no. of times

(2)

- Vanilla model was sometimes simply adding matras when it was really not required, attention was able to correct it.

Example :- input -> survin | correct output -> सुरवीन

vanilla prediction -> सर्विन | attention  
prediction-> सुरवीन

Example :- input -> dablyoopeedee | correct output ->  
डब्ल्यूपीडी

vanilla prediction -> दब्ल्यूपीडी | attention  
prediction-> डब्ल्यूपीडी

- Vowel sounds are very difficult to predict correctly. We can see that both in 'survin' and 'dablyoopeedee', the vowel sound caused the problem.
- Attention helps to understand the context of the sound with respect to whole word. Hence it is able to learn the correct context of a vowel sound. Vanilla is incapable of handling such contexts.

3)

- Vanilla model makes mistake in half characters of hindi, which attention was able to correct

Example :- input -> phabti | correct output -> फब्ती

vanilla prediction -> फाबती | attention  
prediction-> फब्ती

Example :- input -> ambikapur | correct output ->  
अम्बिकापुर

vanilla prediction -> अंबिकापुर | attention  
prediction-> अम्बिकापुर

- Attention mechanisms helps in tasks involving alignment. Half characters are basically alignment problem of how the half char aligns with the neighboring characters. Vanilla model mostly fails in this case since it can't learn in both direction at the same time.

4)

- Vanilla model makes mistake in matras of hindi, which attention was able to correct
- these are the matras like single matra and double matra as seen in first example

Example :- input -> paints | correct output -> पेंट्स

vanilla prediction -> पैट्स | attention

prediction-> पेंट्स

5)

- Vanilla Unable To work Properly For Matras. The consonants which have similar pronunciation as a vowel is predicted wrong by vanilla but corrected by attention

Example :- input -> ambanni | correct output -> अंबानी

vanilla prediction -> अमबानी | attention

prediction-> अंबानी

6)

- Added an extra | Matras wrongly, but attention is able to correct. So, vanilla was wrongly giving more focus on the vowels at the beginning but attention corrected this.

Example :- input -> agavai | correct output -> अगवाई

vanilla prediction -> आगवाई | attention

prediction-> अगवाई

7)

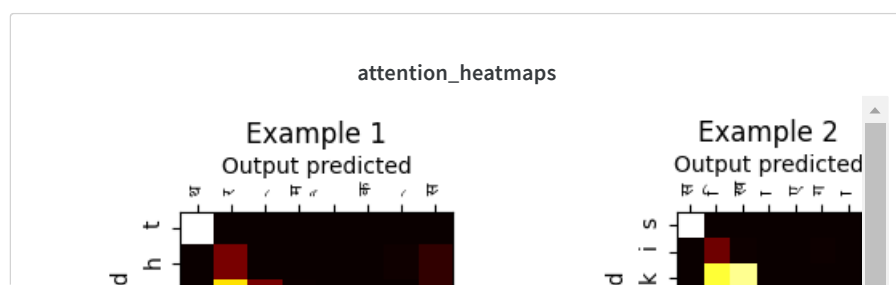
- Added an extra | Matras wrongly, but attention is able to correct. So, vanilla was wrongly giving more focus on the vowels at the beginning but attention corrected this.

Example :- input -> bajuon | correct output -> बाजुओं

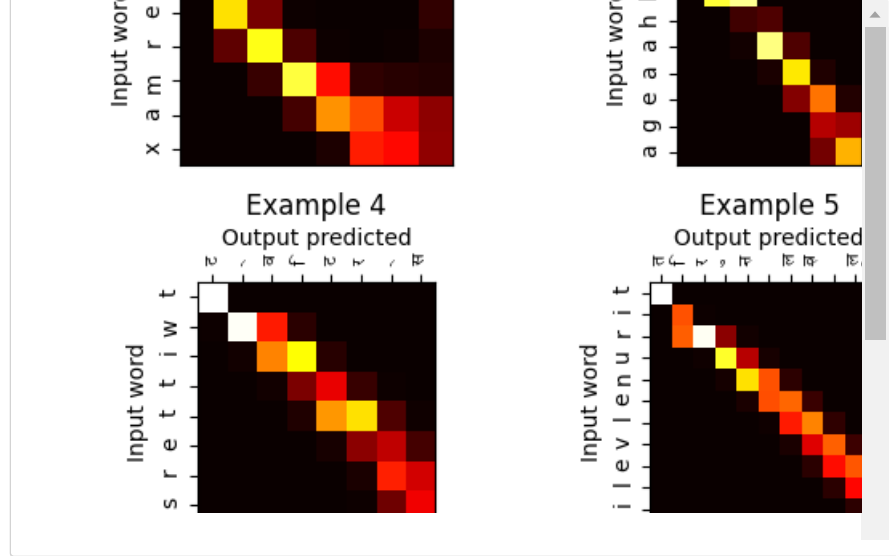
vanilla prediction -> बाजून | attention

prediction-> बाजुओं

d) Attention Heatmaps for 10 inputs (drawn for 12)







### Question 7 (10 Marks)

Paste a link to your GitHub Link

Link : [https://github.com/pankajrawat9075/CS6910\\_assignment\\_3](https://github.com/pankajrawat9075/CS6910_assignment_3)

Readme file is present in the Github repository, which contains all steps to run sweeps and do other important tasks.

### Self-Declaration

I, Pankaj Singh Rawat(Roll no: CS22M062), swear on my honour that I have written the code and the report by myself and have not copied it from the internet or other students.

Created with ❤ on Weights & Biases.

[https://wandb.ai/iitmadrass/CS6910\\_Assignment\\_3/reports/CS6910-Assignment-3-Report-Vmldzo0MzQyNDk5](https://wandb.ai/iitmadrass/CS6910_Assignment_3/reports/CS6910-Assignment-3-Report-Vmldzo0MzQyNDk5)