

Share



CS6910 - Assignment 3

Use recurrent neural networks to build a transliteration system.

[Bibhuti Majhi cs22m031](#)

▼ Instructions

- The goal of this assignment is threefold: (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
- Discussions with other students is encouraged.
- You must use `Python` for your implementation.
- You can use any and all packages from `PyTorch` or `PyTorch-Lightning`. NO OTHER DL library, such as `TensorFlow` or `Keras` is allowed.
- Please confirm with the TAs before using any new external library. BTW, you may want to explore `PyTorch-Lightning` as it includes `fp16` mixed-precision training, wandb integration, and many other black boxes eliminating the need for boiler-plate code. Also, do look out for `PyTorch2.0`.
- You can run the code in a jupyter notebook on Colab/Kaggle 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 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 must also 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 a sample of the [Aksharantar dataset](#) released by [AI4Bharat](#). 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 (how 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 realize, this is the problem of mapping a sequence of characters in one language to a sequence of characters in another. 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 a sequence of **characters** here).

Read this [blog](#) to understand how to build neural sequence-to-sequence models.

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 , the encoder and decoder have 1 layer each, the hidden cell state is k for both the encoder and decoder, and the length of the input and output sequence is the same, i.e., T , the size of the vocabulary is the same for the source and target language, i.e., V)

Answer:

- Sizes Assumed:
 - Embedding size. = m
 - Hidden_size = k
 - no_of_layer = 1 (both for encoder and decoder)
 - length of input and output sequences = T
 - Vocabulary size for input and target languages = V
- $x_i = \text{Embedding}(I_i)$
- $s_i = \sigma(Ux_i + Ws_{i-1} + b)$
- $y_i = \text{Softmax}(V's_i + c)$

- Dimensions each components:
- $I_i = (1, 1)$ (Index representing one character)
- $E = (m, V)$ (Embedding matrix)
- $U = (k, m)$
- $W = (k, k)$
- $s_{i-1} = (k, 1)$
- $b = (k, 1)$
- $v' = (k, k)$
- $s_i = (k, 1)$
- $c = (k, 1)$
- $x_i = (m, 1)$

Total computation in Encoder :

- $s_i = \sigma(Ux_i + Ws_{i-1} + b)$
- In this Equation two operation are happening
 - Multiplication = $O(km + k^2)$
 - Addition = $O(2k)$
- Total computation till now = $O(km + k^2 + 2k)$
- Embedding(I_i) = $O(mV)$
- Sigmoid operation = $O(k)$
- Total computation for encoder in each timestep = $O(mk + k^2 + mV + 3k)$
- Total computation for encoder for T timestep = $O(Tmk + Tmk^2 + TmV + Tk)$

Total computation in Decoder:

- $s_i = \sigma(Ux_i + Ws_{i-1} + b)$
- In this Equation two operation are happening
 - Multiplication = $O(km + k^2)$
 - Addition = $O(2k)$
- Total computation till now = $O(km + k^2 + 2k)$
- Embedding(I_i) = $O(mV)$
- softmax = $O(V)$
- $y_i = \text{softmax}(V's_i + c)$ so computing y_i takes
- $y_i = O(Vk + V)$
- Total computation for Decoder in each timestep = $O(mk + k^2 + mV + Vk + 2V + 2k)$
- Total computation for Decoder in T timestep = $O(Tmk + Tk^2 + TmV + TVk + TV + Tk)$

$$\begin{aligned}\text{Total Computations} &= \text{Computations in Encoder} + \text{Computation in Decoder} \\ &= O(Tmk + TmV + Tk^2 + TVk + TV + Tk)\end{aligned}$$

(b) What is the total number of parameters in your network? (assume that the input embedding size is m , the encoder and decoder have 1 layer each, the hidden cell state is k for both the encoder and decoder, and the length of the input and output sequence is the same, i.e., T , the size of the vocabulary is the same for the source and target language, i.e., V)

Answer:

Encoder :

- Embedding matrix = (m, V)
- $U = (k, m)$
- $W = (k, k)$
- $b = (k, 1)$
- $V' = (k, k)$
- $c = (k, 1)$
- Total parameters in Encoder = $O(mV + km + 2K^2 + 2k)$

Decoder:

- Embedding matrix = (m, V)
- $U = (k, m)$
- $W = (k, k)$
- $b = (k, 1)$
- $V' = (k, k)$
- $c = (k, 1)$
- Total parameters in Decoder = $O(mV + km + k^2 + Vk + V + k)$

$$\text{Total number of Parameters} = O(mV + km + k^2 + k + Vk + V)$$

Question 2 (10 Marks)

You will now train your model using any one language from the [Aksharantar dataset](#) (I would suggest picking a language that you can read so that it is easy to analyze the errors). Use the standard `train`, `dev`, `test` set from the folder `aksharantar_sampled/hin` (replace `hin` with the language of your choice)

BTW, you should read up on how NLG models operate in inference mode, and how it is different from training. This [blog](#) might help you with it.

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

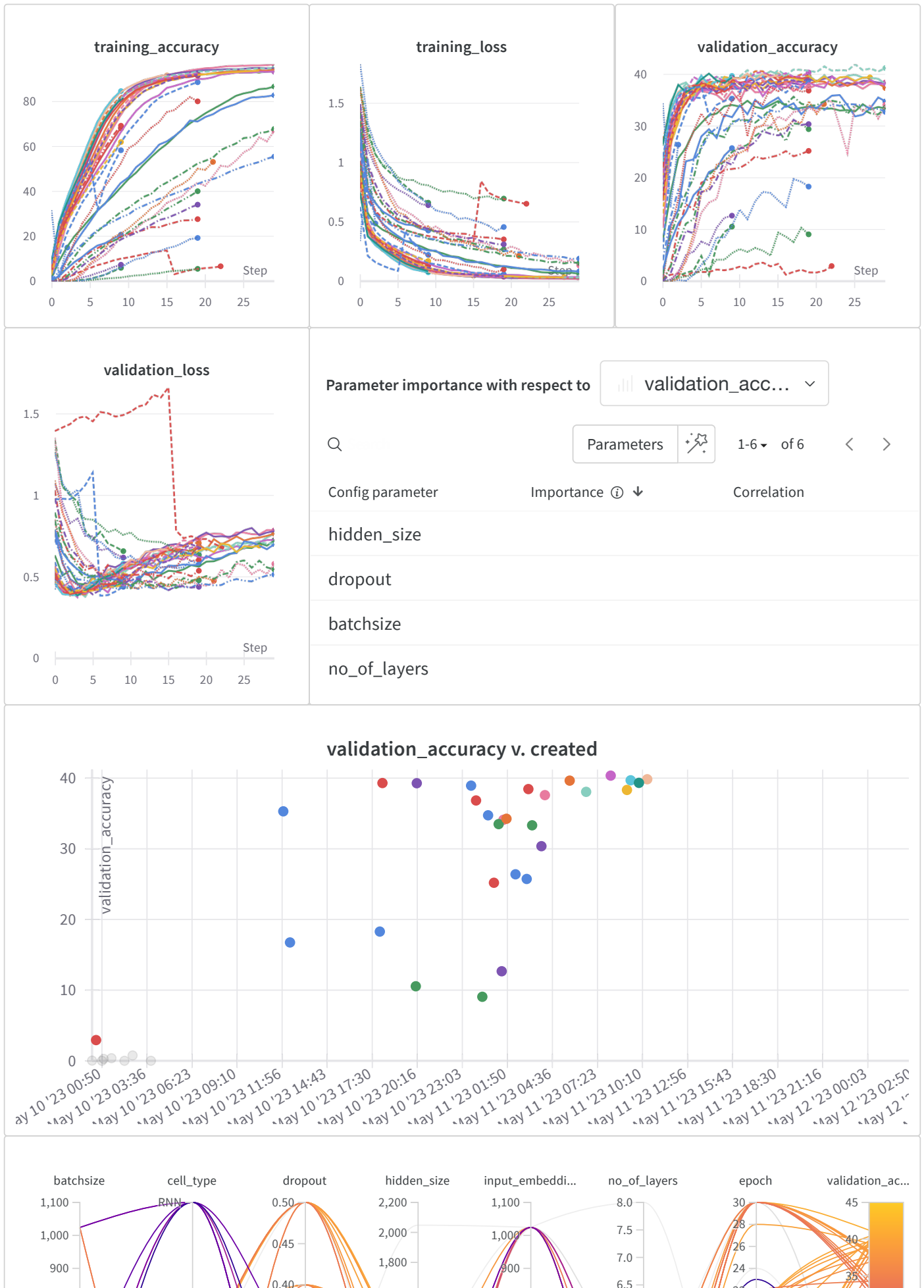
- input embedding size: 16, 32, 64, 256, ...
- number of encoder layers: 1, 2, 3
- number of decoder layers: 1, 2, 3
- hidden layer size: 16, 32, 64, 256, ...
- cell type: RNN, GRU, LSTM
- bidirectional: Yes, No
- dropout: 0.2, 0.3 (btw, where will you add dropout? You should read up a bit on this)
- beam search in decoder with different beam sizes:

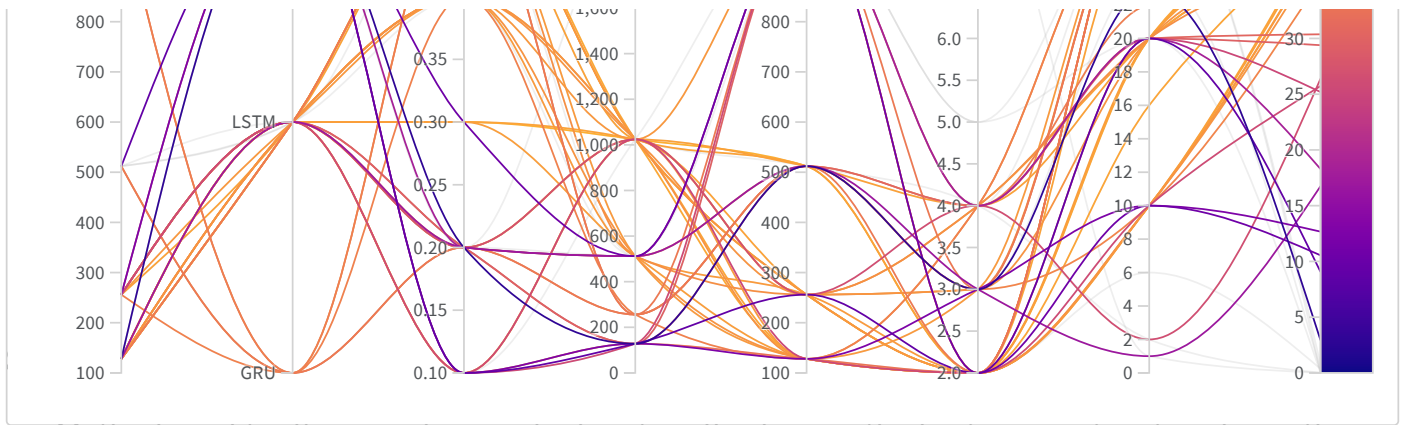
Based on your sweep, please paste the following plots, which are automatically generated by wandb:

- 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 swept 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.

```
sweep_configuration = {
    'method' : 'bayes',
    'metric' : { 'goal' : 'maximize',
    'name' : 'validation_accuracy'},
    'parameters':{
        'batchsize' : {'values' : [64,128,256,512,1024]},
        'input_embedding_size' : {'values' : [128,256,512,1024]},
        'no_of_layers' : {'values' : [1,2,3,4,5,6,7,8]},
        'hidden_size' : {'values' : [128,256,512,1024]},
        'cell_type' : {'values' : ['RNN','GRU','LSTM']},
        'bidirectional' : {'values' : ['Yes']},
        'dropout' : {'values' : [0.1,0.2,0.3,0.4,0.5]},
        'epochs' : {'values' : [10,20,30]}
    }
}
```





- Method used for Sweep is bayes which optimally chooses the best parameters to achieve the required goal.
- Metric used is Maximize which denotes the important parameter and the goal to maximize or minimize it.
- In my sweep configuration
 - Goal : Maximize
 - Name : Validation Accuracy

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 are true. I just wrote some random comments that came to my mind)

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

Answer:

- First ran sweeps with a broad range of parameters like , batchsize = 16,32,64,.....
- Observed that small batchsize takes a lot of time to converge and tends to overfit the training data.
- Decided to run sweeps with only batchsize ≥ 128
- Observed that greater hidden_size and greater embedding_size captures better information about the English words resulting in better validation accuracy.
- Decided to run sweeps with only hidden size ≥ 256 and embedding_size ≥ 256
- Generally increasing the no of layers in the model increases its complexity resulting in better accuracy and also overfitting of training data.
- So decided to reduce the no of layers to 1,2,3,4 only to keep a balance between better accuracy and overfitting.

- Performed sweeps with LSTM,GRU and RNN and observed that LSTM performs better than both RNN and GRU.
- so decided to run sweeps having cell_type = LSTM.
- Greater the no of epochs greater the training accuracy and it also affects the validation accuracy similarly.
- But at a point of time the model starts to overfit the training data and making the validation loss to increase and the validation accuracy to remain plateau.
- Also more epochs leads to more time to converge without any progress in validation accuracy.
- so best no of epochs must be around 20 .
- Now with higher embedding size, we need higher dropout probability to better regularization for unseen data.
- keeping dropout values to 0.1 to 0.5 only cause increasing dropout may also lead to loss of information .
- While researching about this project found out that ADAM optimizer is best suited for NLP tasks.
- So only ran sweeps with ADAM and learning rate = 0.001
- Tried with lesser learning rate than 0.001 and found it to take more time to converge.
- Tried with Greater learning rate than 0.001 and found it to overshoot and cannot find the optimal value.
- so decided to keep learning rate = 0.001.
-

▼ Question 4 (10 Marks)

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

(a) Use the best model from your sweep and report the accuracy on the test set (the output is correct only if it exactly matches the reference output).

- Best Model Configuration for Without attention:
 - Batchsize = 128
 - Hidden_size = 1024
 - Embedding_size = 128
 - Dropout = 0.5
 - Cell_type = LSTM
 - Epochs = 20
 - no_of_layers = 2
 - Bidirectional = True

- Train Accuracy = 92.322265625
- Validation Accuracy = 40.3564453125
- Test Accuracy = 37.744140625

(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.

```
runs.summary["result"]
```

	English	Original	Predicted
19			
20			
21			
22			
23			
24			

← < - 24 of 4096 > →
Export as CSV Columns... Reset Table

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

Answer:

- The model tends to have difficulty in predicting Transliteration of words with many pronunciation like

kaarniyaan

कारनियां

कारनियाँ

latakakar

लटककर

लताकाकर

- This is happening due to the fact that English Language doesn't really capture the difference between many Hindi words like

Take टेक

Talk टॉक

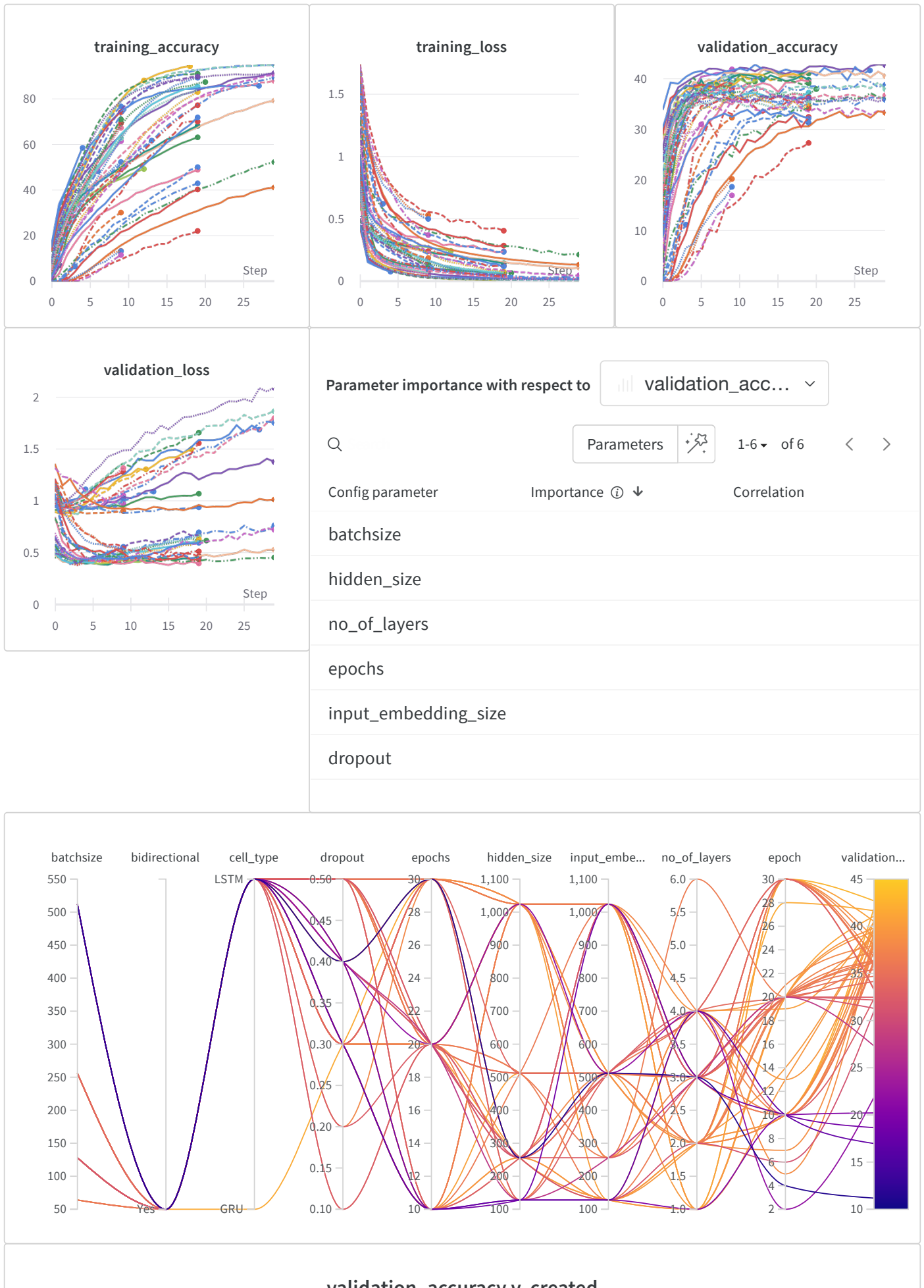
- Both take and talk have 'Ta' in it but one is pronounced as ----- and other is pronounced as -----.
- Also Longer words tend to have more errors than smaller words
- This is because the decoder has information about the English word through the initial hidden/cell state at timestep 1 and this information gets lost throughout the process of decoding .
- To solve this I have given the Original Hidden from Encoder as an input to every timestep of the decoder.

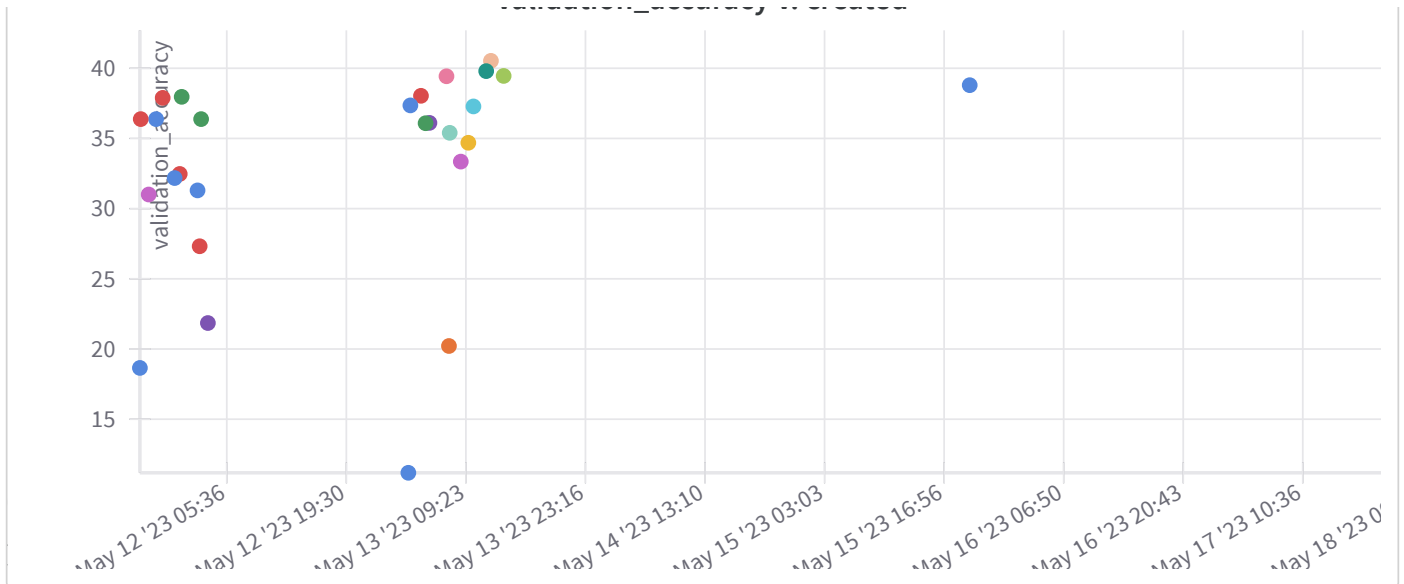
▼ Question 5 (20 Marks)

Add an attention network to your base 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 the appropriate plots below.

```
sweep_configuration = {
    'method' : 'bayes',
    'metric' : { 'goal' : 'maximize',
    'name' : 'validation_accuracy'},
    'parameters':{
        'batchsize' : {'values' : [64, 128, 256, 512, 1024]},
        'input_embedding_size' : {'values' : [128, 256, 512, 1024]},
        'no_of_layers' : {'values' : [1, 2, 3, 4]},
        'hidden_size' : {'values' : [128, 256, 512, 1024]},
        'cell_type' : {'values' : ['RNN', 'GRU', 'LSTM']},
        'bidirectional' : {'values' : ['Yes']},
        'dropout' : {'values' : [0.1, 0.2, 0.3, 0.4, 0.5]},
        'epochs' : {'values' : [10, 20, 30]}
    }
}
```





- Best Model Configuration for Attention:

- Batchsize = 512
- Hidden_size = 1024
- Embedding_size = 1024
- Dropout = 0.4
- Cell_type = LSTM
- Epochs = 30
- no_of_layers = 1

- Train Accuracy = 91.1875

- Validation Accuracy = 42.7001953125

- Test Accuracy = 40.0634765625

(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 that were predicted incorrectly by your best seq2seq model are predicted correctly by this model)

- Yes attention-based model performs better than the vanilla model
- Test Accuracy in attention = ~ 40%
- Test Accuracy in Vanilla = ~37%
- main reasons are :
 - Attention model pays attention to the input characters that are relevant in computing the target characters.
 - Attention model incorporates
- Attention model makes less error w.r.t vanilla model in case of error due to vowels.
- Attention model and vanilla model makes same kind of error in case of consonants.
- Some of the words Predicted correctly by Attention model that had an error in vanilla model:

तिरुनेलवेली

तिरुनल्लेलवी

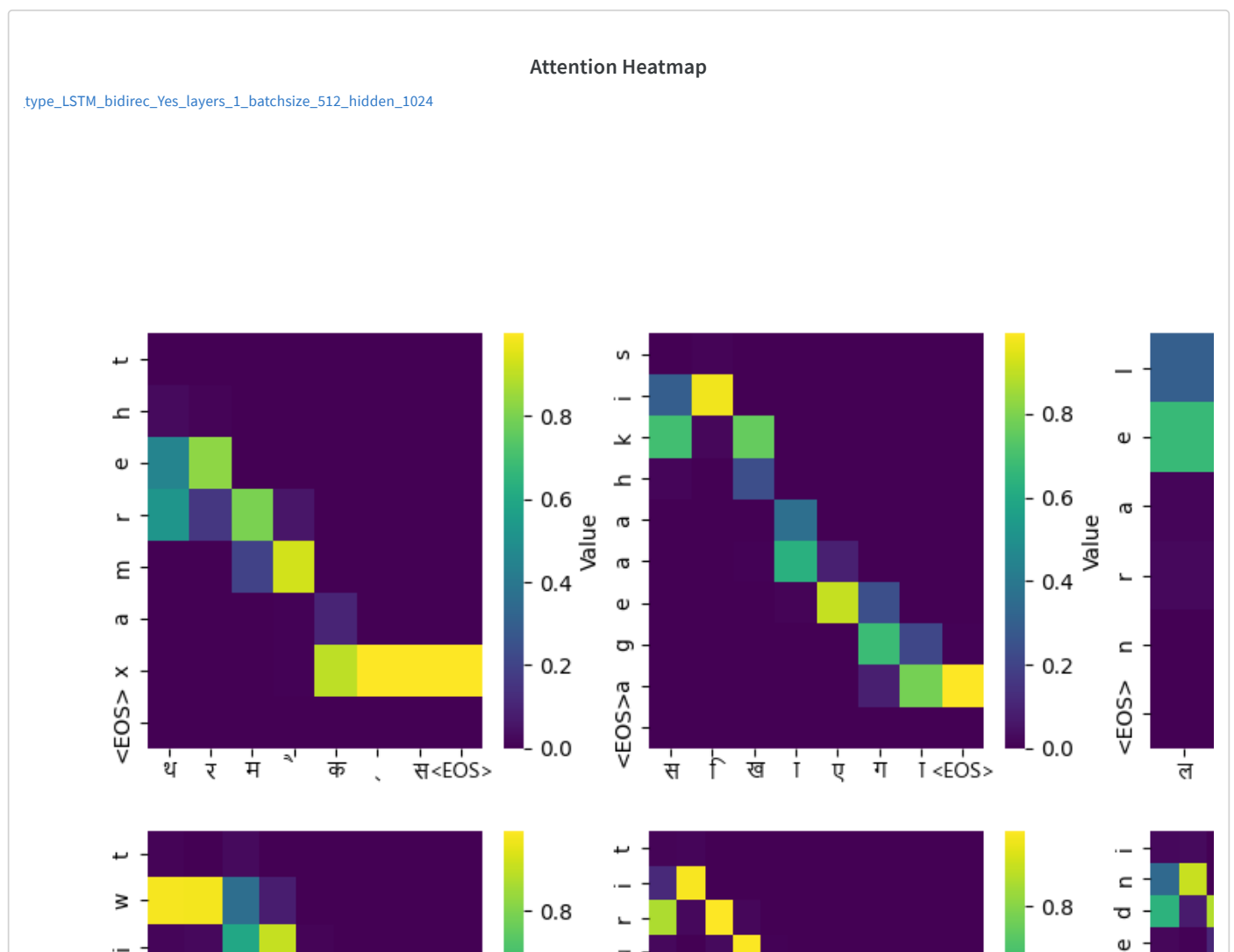
Predictions by Vanilla Model

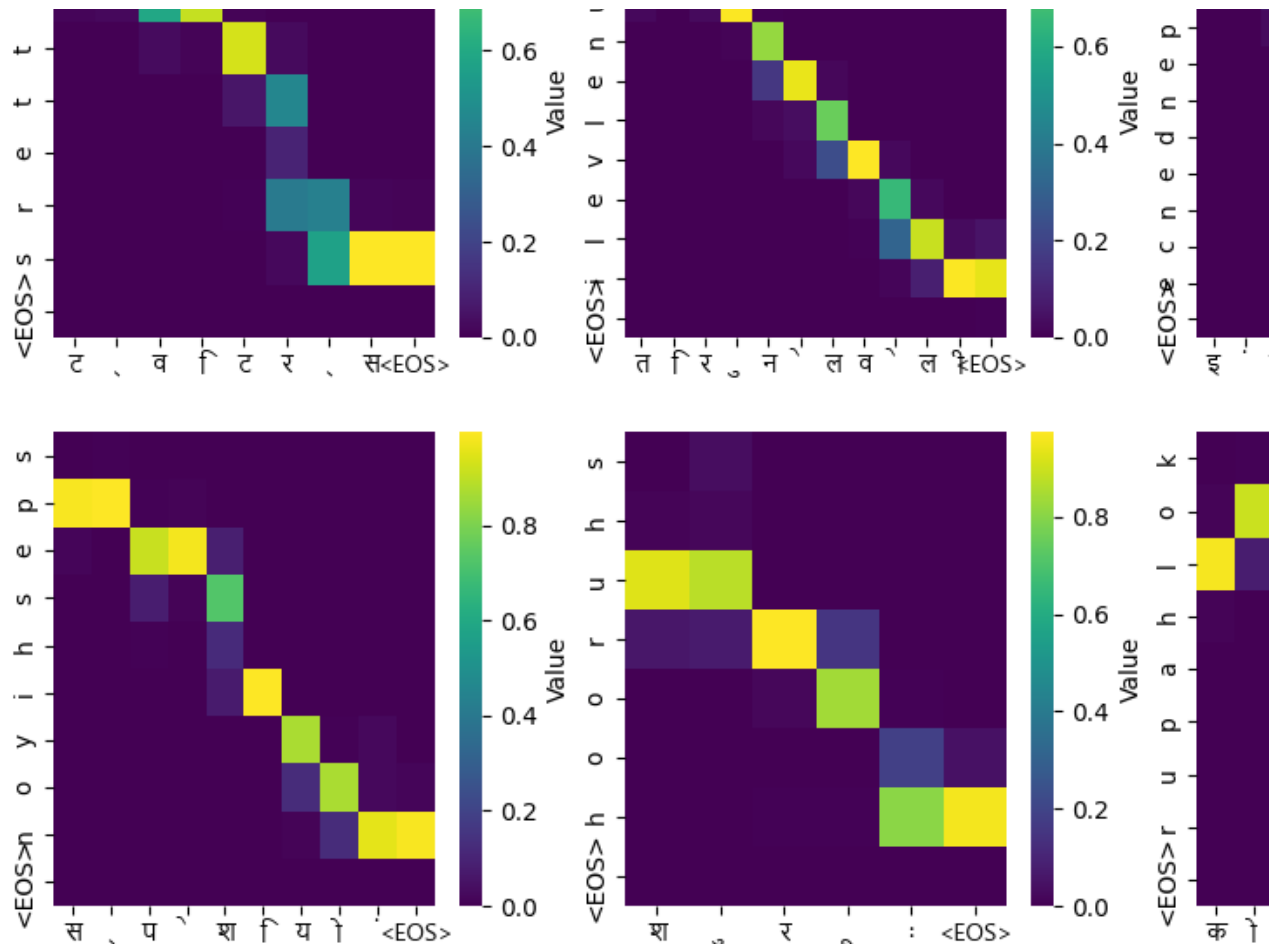
तिरुनेलवेली	तिरुनेलवेली
-------------	-------------

Predictions by Attention Model

- I observed that Attention model has improved predictions on Long length words.

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





▼ (UNGRADED, OPTIONAL) Question 6 (0 Marks)

This is a challenging question, and most of you will find it hard. Note that this question has no marks and will not be graded. This is only for students looking for a challenge and wanting to get something more out of the course.

I like the visualization in the figure captioned "Connectivity" in this [article](#). Make a similar visualization for your model. For some starter code, please look at this [blog](#). 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?

▼ Question 7 (10 Marks)

Paste a link to your GitHub Link

Example: https://github.com/<user-id>/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 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.

link



Github link

Githun link

▼ (UNGRADED, OPTIONAL) Question 8 (0 Marks)


Note that this question has no marks and will not be graded. This is only for students looking for a challenge and wanting to get something more out of the course.

Your task is to finetune the GPT2 model to generate lyrics for English songs. You can refer to this [blog](#) and follow the steps there. This blog shows how to finetune the GPT2 model to generate headlines for financial articles. Instead of headlines, you will use lyrics so you may find the following datasets useful for training: [dataset1](#), [dataset2](#)

At test time, you will give it a prompt: `I love Deep Learning` and it should complete the song based on this prompt :-) Paste the generated song in a block below!

▼ Self-Declaration

I Bibhuti Majhi (Roll no: CS22M031), 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/cs22m031/CS6910_DLAssignment3/reports/CS6910-Assignment-3--Vmlldzo0MzYwMTEx