Assignment 3

Use recurrent neural networks and transformers to build a transliteration system.

Ravish Kumar

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 (iv) understand the importance of Transformers in the context of machine transliteration and NLP in general.
- Discussions with other students are 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 in Google form.
- 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)

(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)

Considering simple RNN model, the number of parameters in a single RNN is as follows:

1. Embedding layer parameters = **Vm**
2. No. of parameters in U = **km** (since U is k x m)
3. No. of parameters in W = **k^2** (since W is k x k)
4. No. of parameters in V = **Vk** (since V is V x k)
5. No. of bias parameters for hidden layer = **k**

6. No. of bias parameters for hidden layer = **V**

An encoder-decoder model with simple **RNN layers** will have the below parameters:

> For the encoder parameters:
> - Embedding layer = Vm
> - Matrix U = km
> - Matrix W = k^2
> - Bias parameters for hidden layer = k

> For the decoder parameters:
> -Embedding layer = Vm
> - Matrix U = km
> - Matrix V = Vk
> - Bias parameters for hidden layer = k
> - Bias parameter for hidden layer = V

Therefore the **total number of parameters** in the **Encoder-decoder** model with a Simple RNN layer is **2Vm + 2Km + 2K^2 + 2k + Vk + V**

Since there are T and T+1 timesteps in encoder and decoder, the total number of computation in this model is **O((2T+1)Vm + (2T+1)km + (2T+1)k^2 + (2T+1)k + (T+1)Vk + (T+1)V)**

For **LSTM layers**,

It's cell has 4 Feedforward Network components,

therefore the total number of parameters in this will be **2Vm + 8km + 8k^2 + 8k + Vk + V**

therefore the number of computations in this model is **O((2T+1)Vm + (2T+1)4km + (2T+1)4k^2 + (2T+1)4k + (T+1)Vk + (T+1)V)**

Similarly for **GRU layers**,

There are 3 feedforward network components,

Number of parameters is **2Vm + 6km + 6k^2 + 6k + Vk + V**

making the number of computation as **O((2T+1)Vm + (2T+1)3km + (2T+1)3k^2 + (2T+1)3k + (T+1)Vk + (T+1)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.

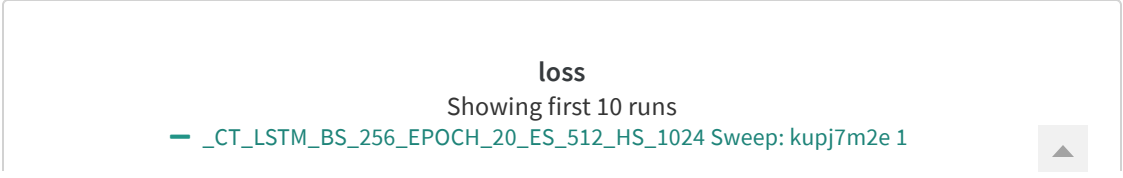
Below is the hyperparameters I used to feed to the wandb.

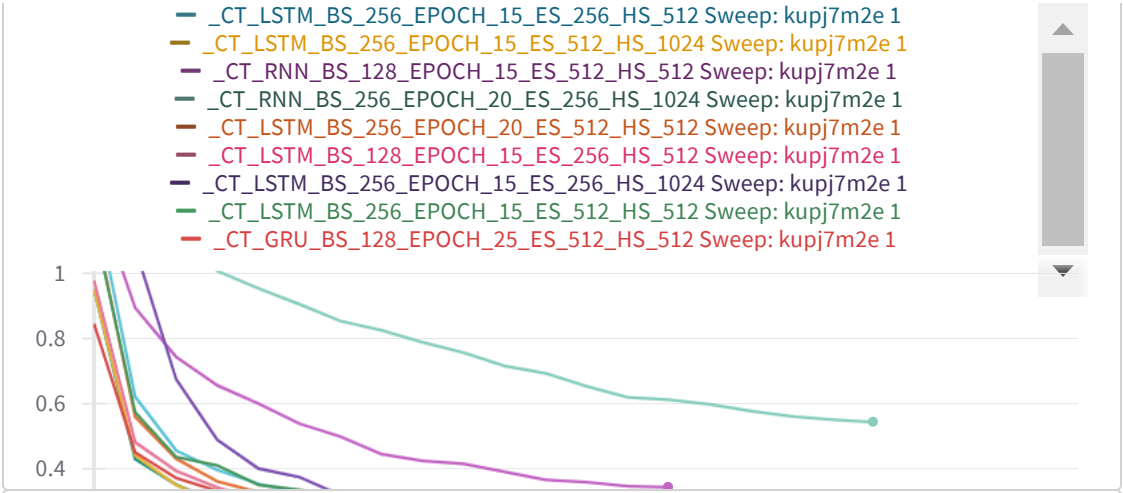
```
config = {
        "project":"Assignment3",
        "method": 'bayes',
        "metric": {
        'name': 'acc',
        'goal': 'maximize'
        },
        'parameters' :{
        "epochs": {"values":[15,20,25]},
        "batchsize": {"values": [64,128,256]},
        "embedding_size": {"values":[256,512,1024]},
        "hidden_size": {"values":[256,512,1024]},
        "encoder_layers": {"values":[2,3,4]},
        "decoder_layers": {"values":[2,3,4]},
        "cell_type": {"values":["LSTM"]},
        "bi_directional":{"values":["Yes","No"]},
        "dropout":{"values":[0.2, 0.3, 0.5]},
        "attention":{"values":["No","Yes"]},
        }
    }
```

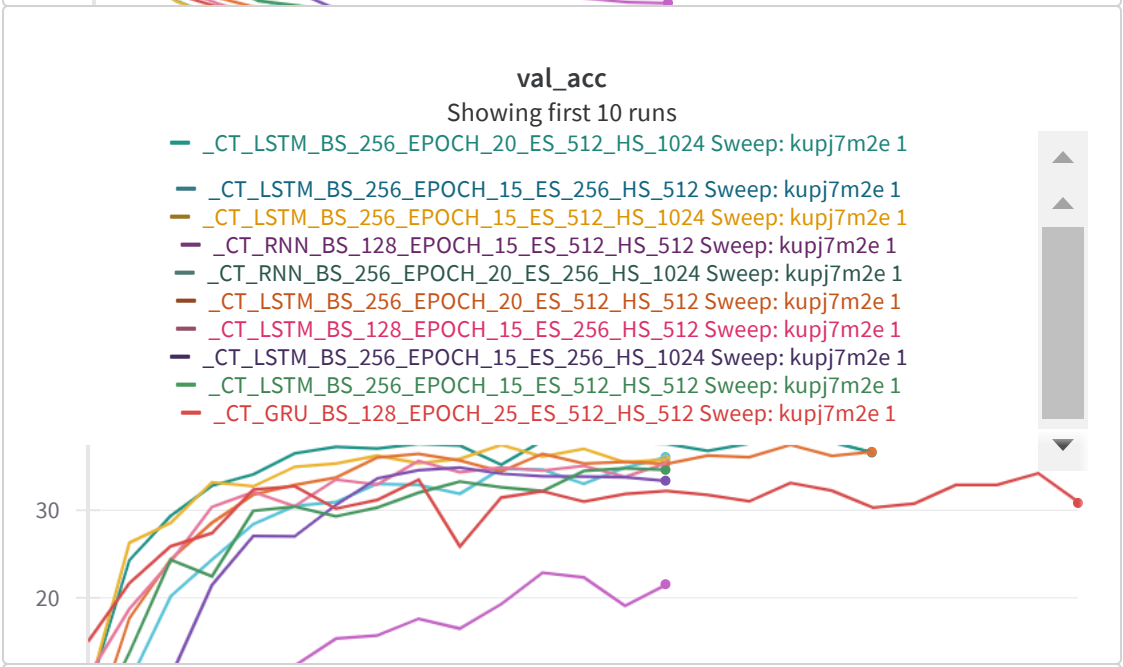
Strategies used for searching the hyperparameters are as follows:

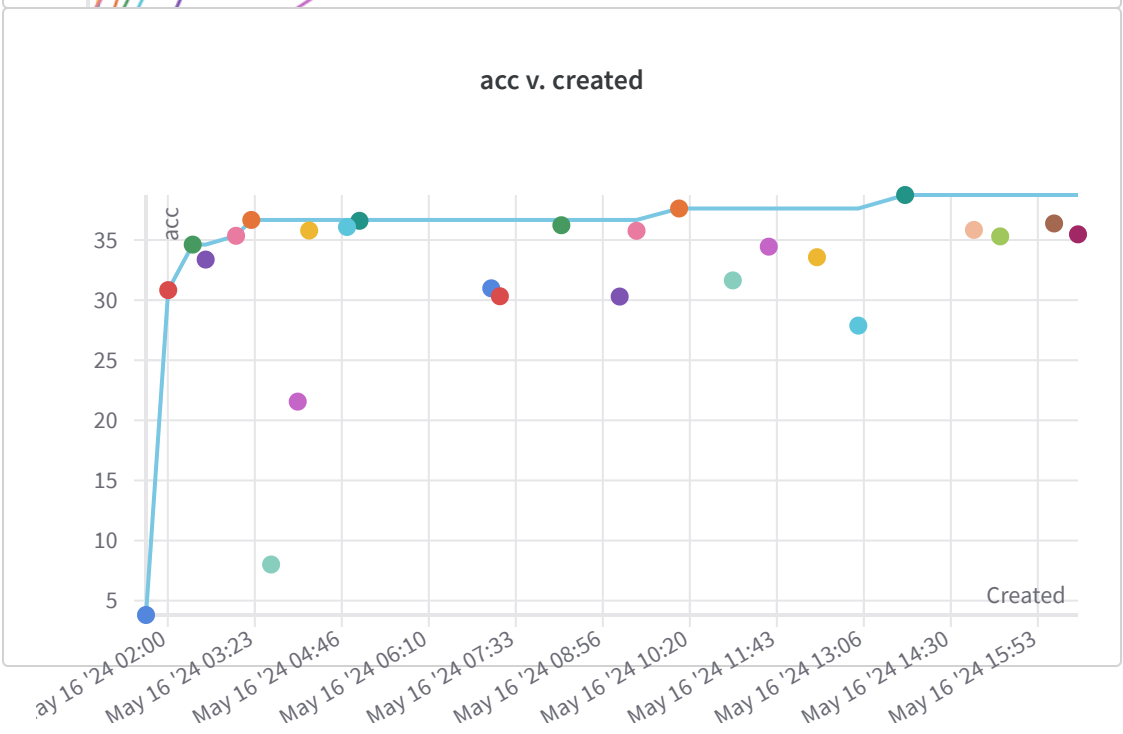
Wandb's bayesian search strategy - It uses gaussian process to model the function and then chooses parameters to optimize the probability of improvement.

Initially I used the config provided here and observed some hyperparameters giving the good results and then only after some runs built the config mentioned above and selected the wandb's bayesian strategy for the better results.




_CT_LSTM_BS_256_EPOCH_15_ES_256_HS_512 Sweep: kupj7m2e 1
_CT_LSTM_BS_256_EPOCH_15_ES_512_HS_1024 Sweep: kupj7m2e 1
_CT_RNN_BS_128_EPOCH_15_ES_512_HS_512 Sweep: kupj7m2e 1
_CT_RNN_BS_256_EPOCH_20_ES_256_HS_1024 Sweep: kupj7m2e 1
_CT_LSTM_BS_256_EPOCH_20_ES_512_HS_512 Sweep: kupj7m2e 1
_CT_LSTM_BS_128_EPOCH_15_ES_256_HS_512 Sweep: kupj7m2e 1
_CT_LSTM_BS_256_EPOCH_15_ES_256_HS_1024 Sweep: kupj7m2e 1
_CT_LSTM_BS_256_EPOCH_15_ES_512_HS_512 Sweep: kupj7m2e 1
_CT_GRU_BS_128_EPOCH_25_ES_512_HS_512 Sweep: kupj7m2e 1
1
0.8
0.6
0.4


val_acc
Showing first 10 runs
_CT_LSTM_BS_256_EPOCH_20_ES_512_HS_1024 Sweep: kupj7m2e 1
_CT_LSTM_BS_256_EPOCH_15_ES_256_HS_512 Sweep: kupj7m2e 1
_CT_LSTM_BS_256_EPOCH_15_ES_512_HS_1024 Sweep: kupj7m2e 1
_CT_RNN_BS_128_EPOCH_15_ES_512_HS_512 Sweep: kupj7m2e 1
_CT_RNN_BS_256_EPOCH_20_ES_256_HS_1024 Sweep: kupj7m2e 1
_CT_LSTM_BS_256_EPOCH_20_ES_512_HS_512 Sweep: kupj7m2e 1
_CT_LSTM_BS_128_EPOCH_15_ES_256_HS_512 Sweep: kupj7m2e 1
_CT_LSTM_BS_256_EPOCH_15_ES_256_HS_1024 Sweep: kupj7m2e 1
_CT_LSTM_BS_256_EPOCH_15_ES_512_HS_512 Sweep: kupj7m2e 1
_CT_GRU_BS_128_EPOCH_25_ES_512_HS_512 Sweep: kupj7m2e 1
30
20


acc v. created
acc
35
30
25
20
15
10
5
Created
May 16 '24 02:00
May 16 '24 03:23
May 16 '24 04:46
May 16 '24 06:10
May 16 '24 07:33
May 16 '24 08:56
May 16 '24 10:20
May 16 '24 11:43
May 16 '24 13:06
May 16 '24 14:30
May 16 '24 15:53

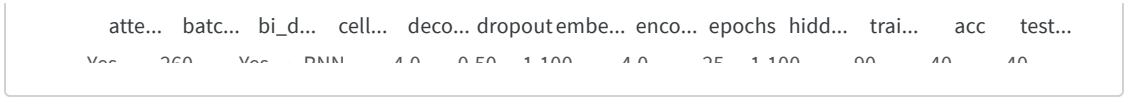
Parameter importance with respect to test_accuracy

Search

Parameters

1-7 of 7

Config parameter	Importance	Correlation
encoder_layers		
batchsize		
dropout		
embedding_size		
epochs		
decoder_layers		
hidden_size		



Question 3 (15 Marks)

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

- RNN 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.

Name: **_CT_LSTM_BS_128_EPOCH_25_ES_256_HS_1024**
attention: **No**
batchsize: **128**
bi_directional: **Yes**
cell_type: **LSTM**
decoder_layers: **2**
dropout: **0.3**
embedding_size: **256**
encoder_layers: **4**
epochs: **25**
hidden_size: **1024**
acc: **36.23**
test_accuracy: **33.325**

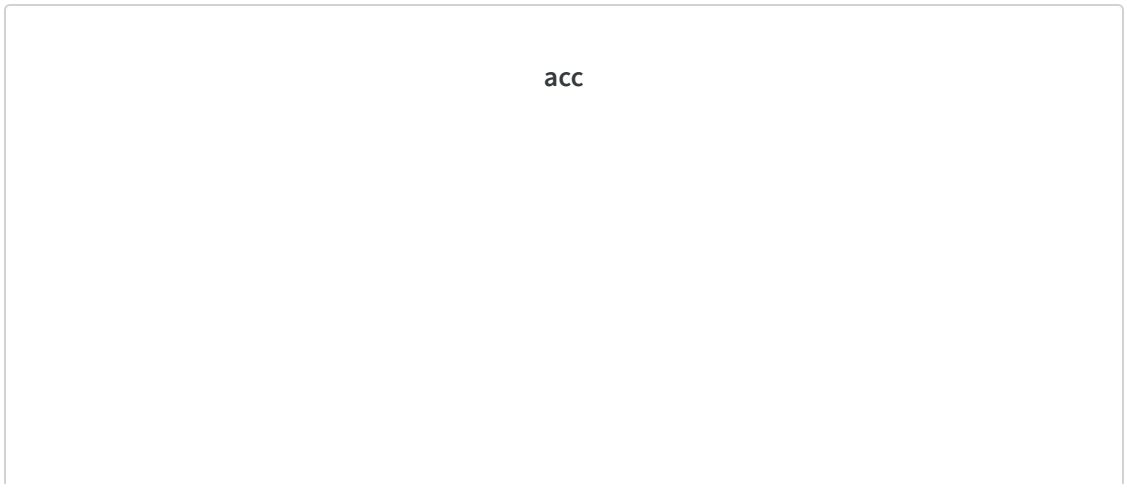
Above is the best sweep configuration achieved with test accuracy of **33.325%** and validation accuracy of **36.23%** without attention.

- LSTM and GRU performance is better than RNN even after adding more layers.
- More number of layers were needed in the encoder and decoder layer in the case of GRU for better performance.
- Adam was able to converge quickly.
- More the batch size, the better was the accuracy.
- It is a good choice to choose the learning rate as a not very small neither too large as in both will take longer time to converge, so 0.001 is a good choice.
- Dropout does help model from overfitting. The dropout size of 0.3 gave good results as can be seen from the parallel coordinate chart.
- For training encoder and decoder with multiple layers took more time than to train for single layer of encoder and decoder.
- More the epochs, more the accuracy. Most of the higher accuracy was achieved for 25 epochs.
- More the hidden layers, more is the accuracy.

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.

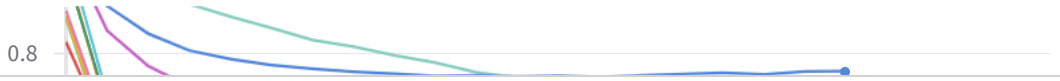


_CT_LSTM_BS_256_EPOCH_20_ES_512_HS_1024 Run set 2

loss

Showing first 10 runs

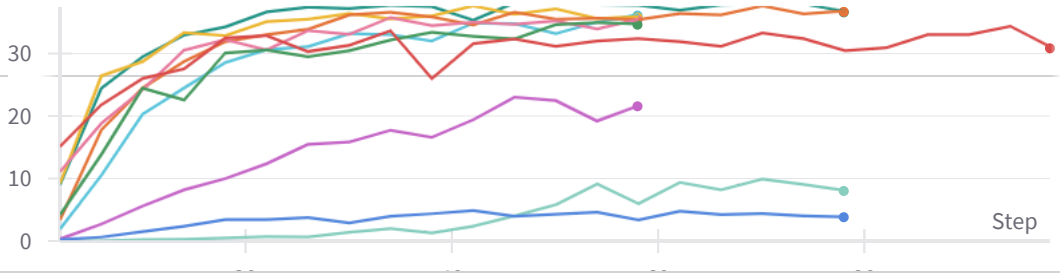
- _CT_LSTM_BS_256_EPOCH_20_ES_512_HS_1024 Run set 2
- _CT_LSTM_BS_256_EPOCH_15_ES_256_HS_512 Run set 2
- _CT_LSTM_BS_256_EPOCH_15_ES_512_HS_1024 Run set 2
- _CT_RNN_BS_128_EPOCH_15_ES_512_HS_512 Run set 2
- _CT_RNN_BS_256_EPOCH_20_ES_256_HS_1024 Run set 2
- _CT_LSTM_BS_256_EPOCH_20_ES_512_HS_512 Run set 2
- _CT_LSTM_BS_128_EPOCH_15_ES_256_HS_512 Run set 2
- _CT_LSTM_BS_256_EPOCH_15_ES_512_HS_512 Run set 2
- _CT_GRU_BS_128_EPOCH_25_ES_512_HS_512 Run set 2
- _CT_RNN_BS_64_EPOCH_20_ES_1024_HS_512 Run set 2

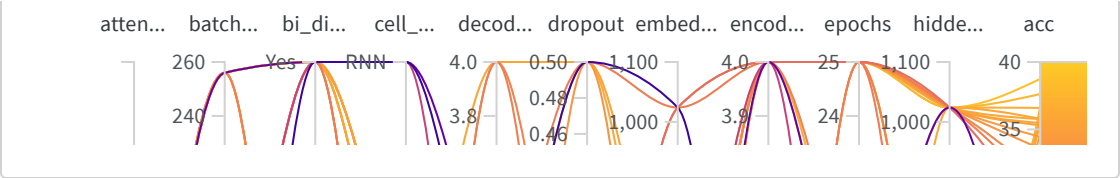


val_acc

Showing first 10 runs

- _CT_LSTM_BS_256_EPOCH_20_ES_512_HS_1024 Run set 2
- _CT_LSTM_BS_256_EPOCH_15_ES_256_HS_512 Run set 2
- _CT_LSTM_BS_256_EPOCH_15_ES_512_HS_1024 Run set 2
- _CT_RNN_BS_128_EPOCH_15_ES_512_HS_512 Run set 2
- _CT_RNN_BS_256_EPOCH_20_ES_256_HS_1024 Run set 2
- _CT_LSTM_BS_256_EPOCH_20_ES_512_HS_512 Run set 2
- _CT_LSTM_BS_128_EPOCH_15_ES_256_HS_512 Run set 2
- _CT_LSTM_BS_256_EPOCH_15_ES_512_HS_512 Run set 2
- _CT_GRU_BS_128_EPOCH_25_ES_512_HS_512 Run set 2
- _CT_RNN_BS_64_EPOCH_20_ES_1024_HS_512 Run set 2





(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.

Name: **_CT_LSTM_BS_256_EPOCH_25_ES_256_HS_1024**
attention: **Yes**
batchsize: **256**
bi_directional: **Yes**
cell_type: **LSTM**
decoder_layers: **2**
dropout: **0.3**
embedding_size: **256**
encoder_layers: **4**
epochs: **25**
hidden_size: **1024**
train_accuracy: **85.49**
acc: **38.745**
test_accuracy: **35.303**

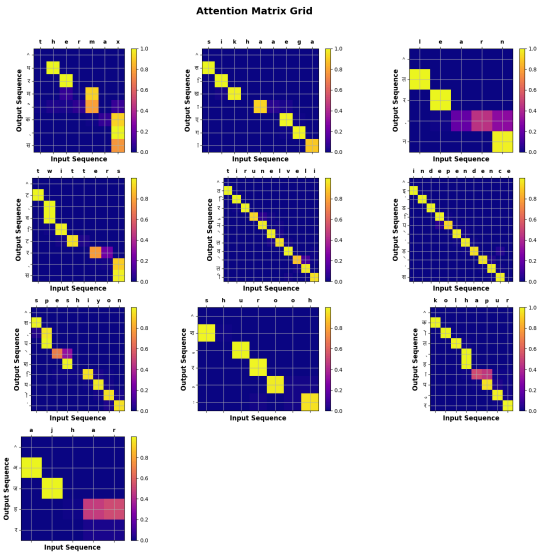
Above is the best configuration with test accuracy of **35.303%** and training accuracy of **85.49%** with attention.

(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 performed better than vanilla model as now it pays more attention to the relevant input words at each timestep, this helps in incorporating the influence of nearby surrounding character to effectively output at each timestep.

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

Attention Images



Question 8 (10 Marks)

Paste a link to your GitHub Link

https://github.com/ravishk17/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.

Self-Declaration

I, Ravish Kumar CS23M055, 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.