Share Comment Star

Assignment3

Use recurrent neural networks to build a transliteration system

Manish Kumar Kumawat ma23m010

Created on May 17 | Last edited on August 18

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)

Answer: Building an RNN-based Seq2Seq Model

(a). (a) Total Number of Computations

Given the parameters:

Input embedding size:m

Hidden state size: k

Sequence length: T

Vocabulary size: V

For each character in the input sequence, the computations involve:

(1)Embedding Layer: For each input character, an embedding lookup, which is $O(1)$.

(2)Encoder RNN: For each time step t in the sequence, the computations involve matrix multiplications and activations:

- Input to hidden:$O(m \times k)$
- Hidden to hidden: $O(k \times k)$
- Total per time step: $O(mk + k^2)$
- Total for T steps: $O(T \times (mk + k^2))$

For the decoder RNN, the computations are similar.

(b)Total Number of Parameters

Parameters are calculated as follows:

(1)Embedding Layer:$V \times m$

(1)Encoder RNN:

- Input to hidden weights: $m \times k$
- Hidden to hidden weights: $k \times k$
- Biases: k
- Total: $mk + k^2 + k$

(3)Decoder RNN:

- Hidden to hidden weights: $k \times k$
- Output weights: $k \times V$
- Biases: $k + V$
- Total: $k^2 + kV + k + V$

Combining both encoder and decoder, total parameters:$V \times m + (mk + k^2 + k) + (k^2 + kV + k + V)$

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: $16, 32, 64, 256, \ldots$
- number of encoder layers: $1, 2, 3$
- number of decoder layers: $1, 2, 3$
- hidden layer size: $16, 32, 64, 256, \ldots$
- 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:

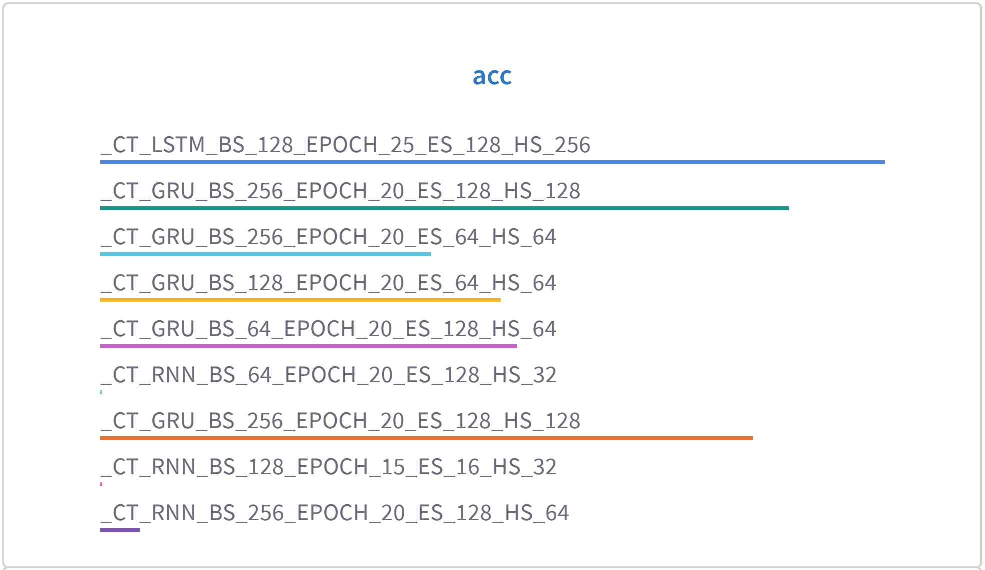
Based on your sweep, please paste the following plots, which are automatically generated by wandb:

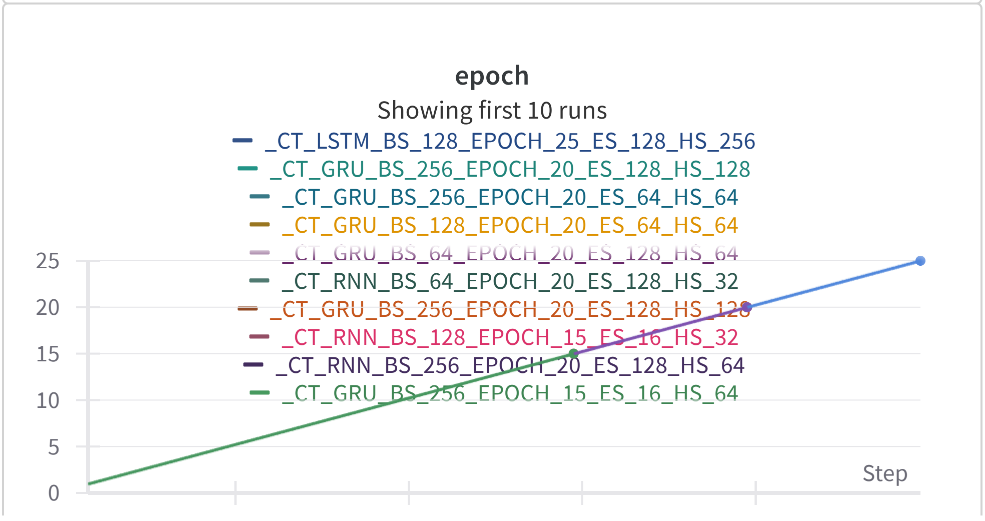
- accuracy v/s created plot (I would like to see the number of experiments you ran to get the best configuration).
- parallel co-ordinates plot
- correlation summary table (to see the correlation of each hyperparameter with the loss/accuracy)

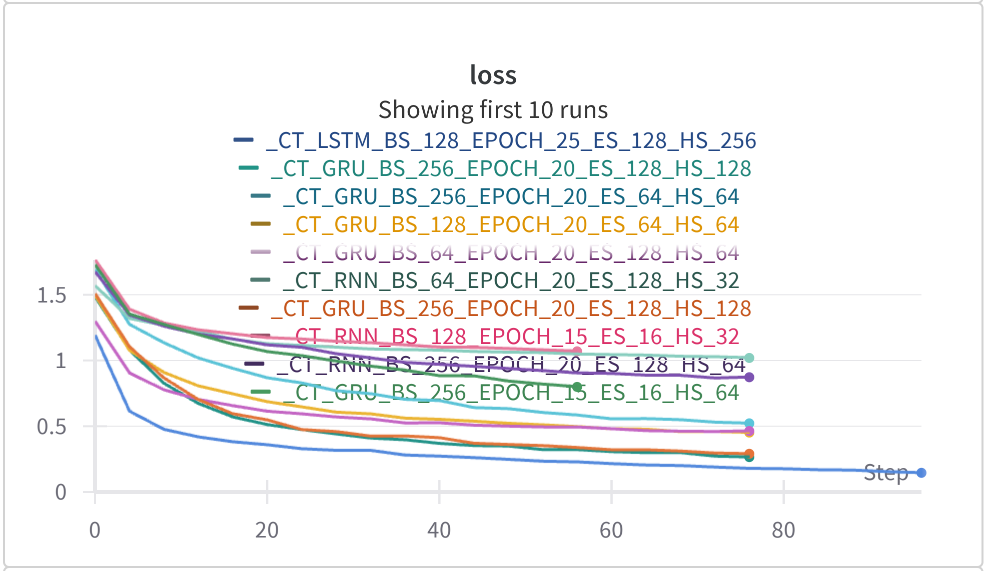
Also, write down the hyperparameters and their values that you swept over. Smart strategies to reduce the number of runs while still achieving a high accuracy would be appreciated. Write down any unique strategy that you tried for efficiently searching the hyperparameters.

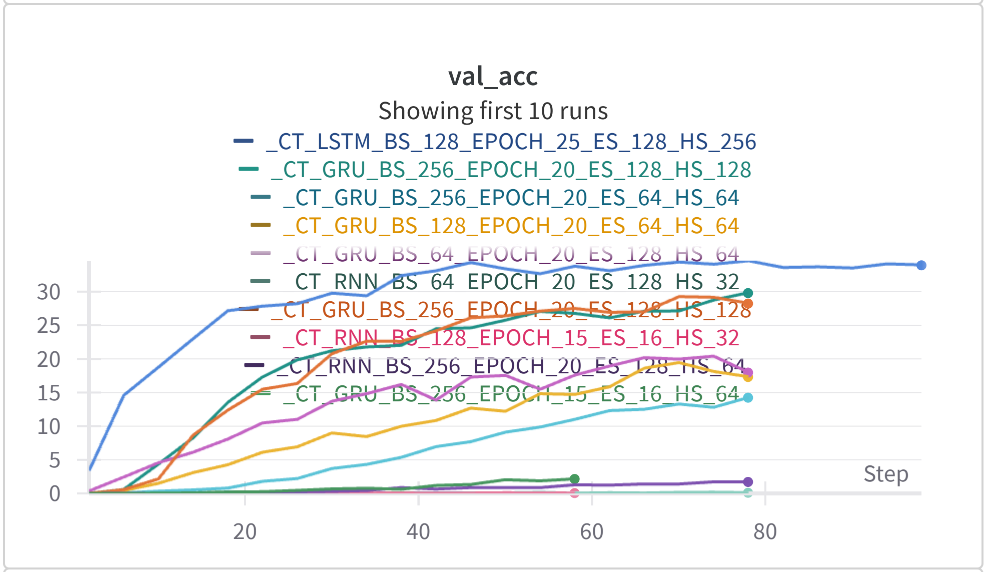
Answer:

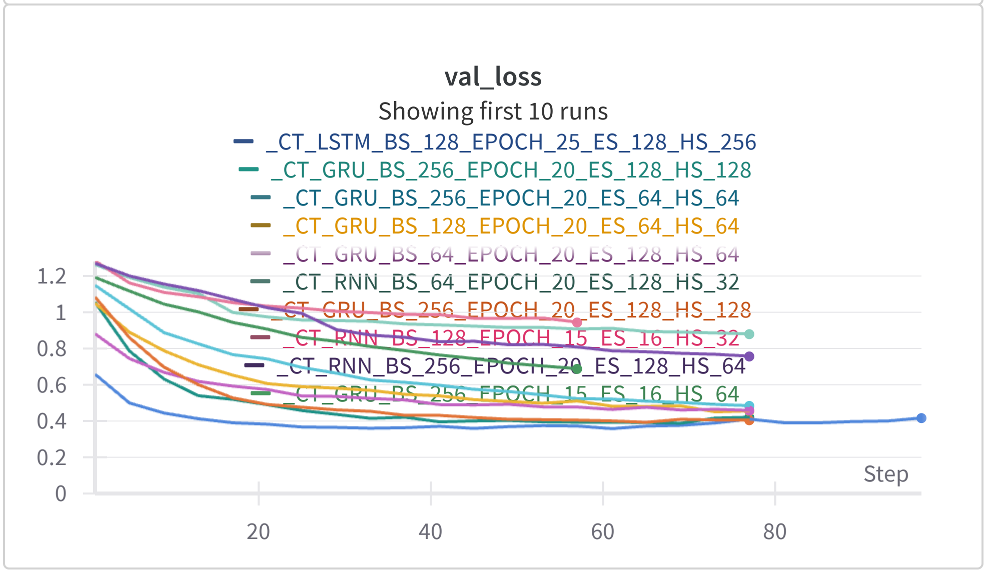
- The following was the hyperparameter space that was swept over.
    - input embedding size: 64, 256, 512, 1024
    - number of encoder layers: 1, 2, 3, 4
    - number of decoder layers: 1, 2, 3, 4
    - hidden layer size: 32, 64, 256, 1024
    - cell type: RNN, GRU, LSTM
    - dropout: 0.1, 0.2, 0.3, 0.4
    - epochs: 15, 20, 25
    - Bi_directional: "Yes", "No"



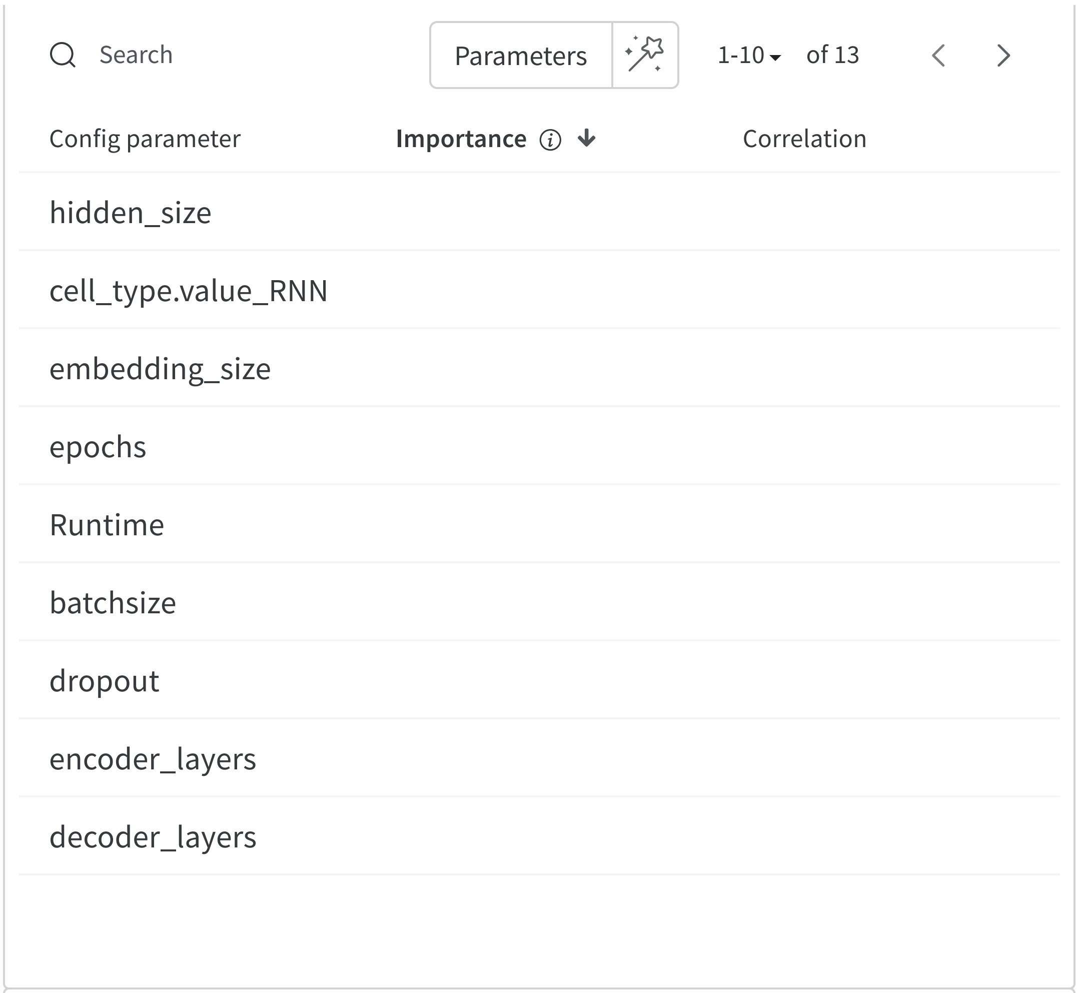






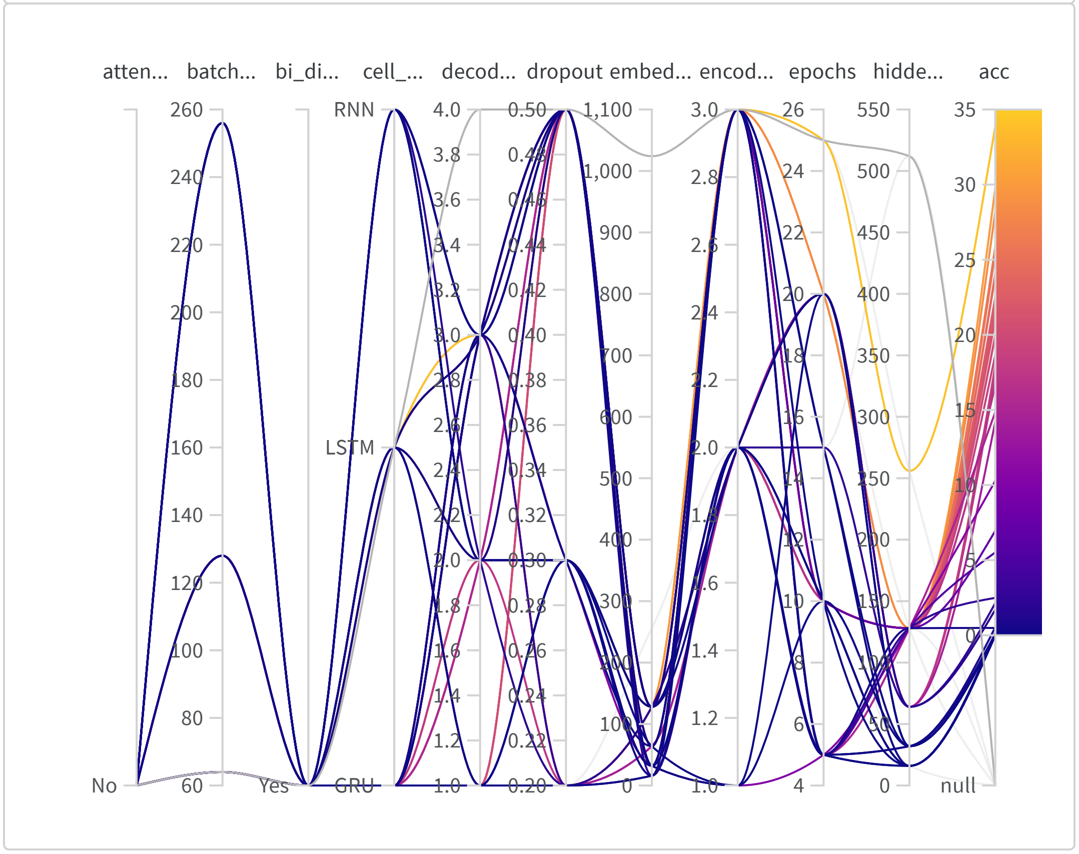


Parameter importance with respect to acc



Search Parameters 1-10 of 13

Config parameter	Importance	Correlation
hidden_size		
cell_type.value_RNN		
embedding_size		
epochs		
Runtime		
batchsize		
dropout		
encoder_layers		
decoder_layers		



Strategy to Reduce the Number of Runs

Random Search: Utilized random search for hyperparameter optimization, which is more efficient than grid search when the number of hyperparameters is large.

Early Stopping: Implemented early stopping in wandb to terminate less promising runs early.

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 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:

best accuracy for validation data is coming 33.91%

Reporting Accuracy of the Best Model

input embedding size: 128

number of encoder layers: 3

number of decoder layers: 3

hidden layer size: 256

cell type: LSTM

dropout: 0.1

epochs: 25

Bi_directional: "Yes"

- RNN performing worse compared to either LSTM or GRU.
- LSTM performance better with a lower number of layers in the encoder and decoder.
- GRU requires more layers in the encoder and decoder for better performance.
- The value of 0.2 gives good performance.
- A dropout value of 0.4 seems high (for most models) and a value of 0 leads to overfitting.
- Optimizers other than Adam failed to converge quickly. Hence the optimizer was fixed as Adam.
- Hidden size is positively correlated with accuracy.
- Higher input embedding size helps in increasing performance by allowing to learn the input with greater contrast.
- The models with higher number of units (i.e., hidden layer units) performs better than the models having the lesser unit size.
- Higher input embedding size gives better the accuracy.
- It is preferred to take Learning rate as the intermediate value (like 1e-2) because the higher (1e-0, 1e-1) and lower (1e-5) learning rates takes more time for convergence.
- Time taken to train a model with multiple encoder and decoder layers is more compared to the model with single encoder and decoder layer.
- epochs are positivel related to the performance of the model and also noticed that 25 epochs provides the better results within the significant amount of time.

Question 4 (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.

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

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

Answer:

Now after applying attention my hyperparameters are:

bi_directional: No

cell_type: LSTM

decoder_layers: 3

dropout: 0.2

embedding_size: 256

encoder_layers: 3

epochs: 25

hidden_size: 256

My best accuracy for validation data is coming 36.74%

Yes the attention-based model perform better than the vanilla model.

Question 5 (10 Marks)

Paste a link to your GitHub Link

https://github.com/manish2021iitd/Deep-Learning

Self-Declaration

I, Manish kumar kumawat(MA23M010), 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.