Autosaved just now

Publish changes

This report contains Markdown blocks. Would you like to convert them into WYSIWYG?

Convert Dismiss

cs20b083's Assignment 3

Use recurrent neural networks to build a transliteration system.

Mantra Trambadia

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)

(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

I have created a `Translator` class which abstracts `Seq2Seq` model and `DataLoader`, it is used to train and evaluate a translator from any of the languages present in Aksharantar dataset.

Note that I have implemented code with support for batchwise computation. We need each input with same length in this case and thus I have used padding character(`_`) and made all of them of `max_size` if batch size is more than 1.

One interesting benefit of padding is that you can forward any number of inputs at a time. So, if batch size more than 1 is used then I will forward all of the data in one single huge batch while computing final accuracy and loss. This saves a lot of time as those computations are expensive.

(a) The total number of computations done by my RNN seq2seq model is as follows:

- Encoder:
  - For each character: Hidden state computation:
    - Embedding computation: Dot-product: V * m
    - From input: Matrix product (1 m @ m k): m * k
    - Recurrent: Matrix product (1 k @ k k): k * k
  - Total computation: T *(*V m + m k + k k)
- Decoder:
  - For each character: Hidden state computation:
    - Embedding previous character: Matrix product (1 V @ V m): V * m
    - [illegible]

        - Embedding to hidden: Matrix product (1 *m* @ *m* k): m ^ k
        - Context computation: Matrix product (1 *k* @ *k* k): k * k
        - Output computation: Matrix product (1 *k* @ *k* V): k * V
    - Total computation: T *(V* m + m *k + k* k + k * V)

Overall, the total number of computations in the network is the sum of the computations in the encoder and decoder:

T *(2mV + 2mk + 2k*k + kV)

(b) The total number of parameters in my RNN seq2seq model is as follows:

- Encoder:
    - Embedding: m * V
    - Input weight: m * k
    - Recurrent weight: k * k
    - Bias vector: k
- Decoder:
    - Embedding previous weight: m * V
    - Input weight: m * k
    - Context weight: k * k
    - Bias vector: k
    - Output weight: k * V

Overall, the total number of parameters in the network is the sum of the parameters in the encoder and decoder:

2mV + 2mk + 2k*k + kV + 2k

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.

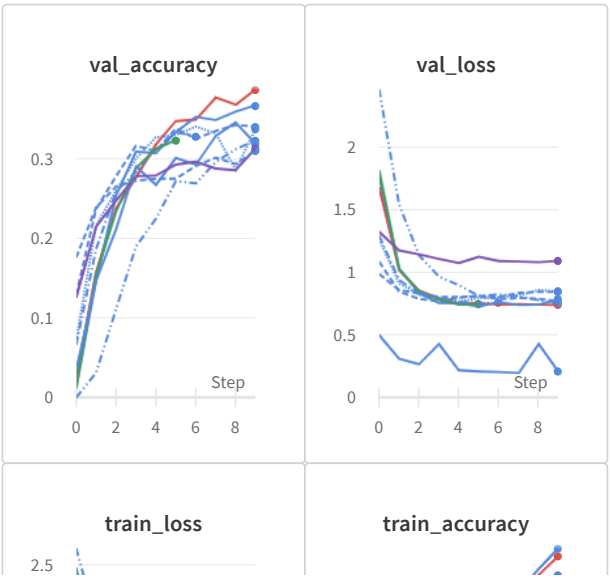
Using the sweep feature in wandb find the best hyperparameter configuration. Here are some suggestions, but you are free to decide which hyperparameters you want to explore

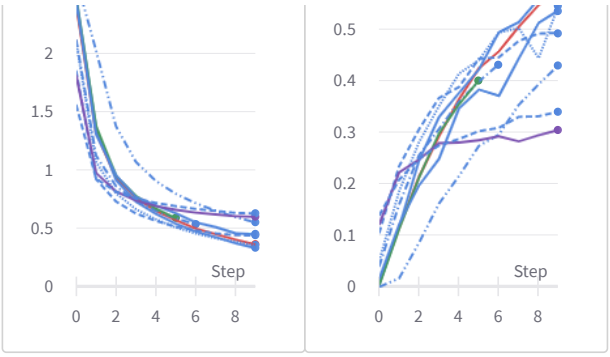
- input embedding size: 16, 32, 64, 256, ...
- number of encoder layers: 1, 2, 3
- number of decoder layers: 1, 2, 3
- hidden layer size: 16, 32, 64, 256, ...
- cell type: RNN, GRU, LSTM
- bidirectional: Yes, No
- dropout: 0.2, 0.3 (btw, where will you add dropout? You should read up a bit on this)
-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

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

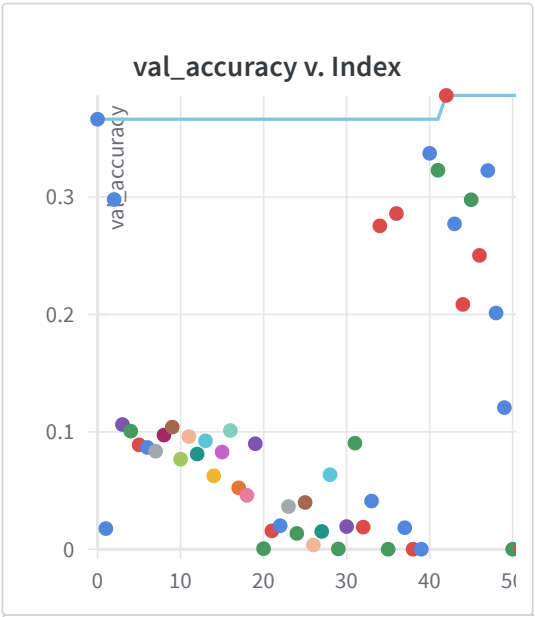
- accuracy v/s created plot (I would like to see the number of experiments you ran to get the best configuration).
- parallel co-ordinates plot
- correlation summary table (to see the correlation of each hyperparameter with the loss/accuracy)

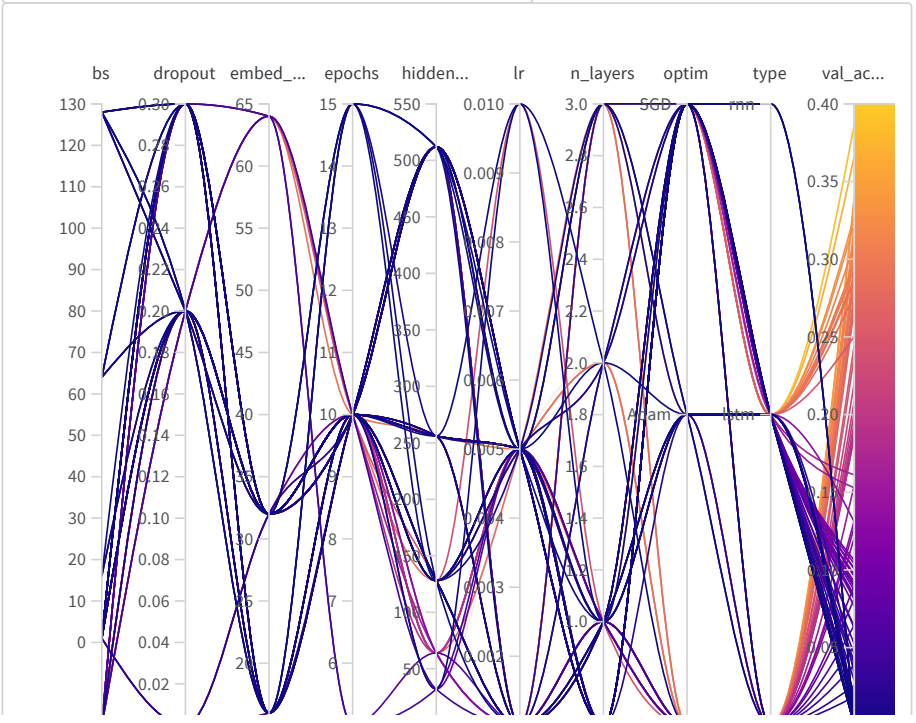
Also, write down the hyperparameters and their values that you swept over. Smart strategies to reduce the number of runs while still achieving a high accuracy would be appreciated. Write down any unique strategy that you tried for efficiently searching the hyperparameters.

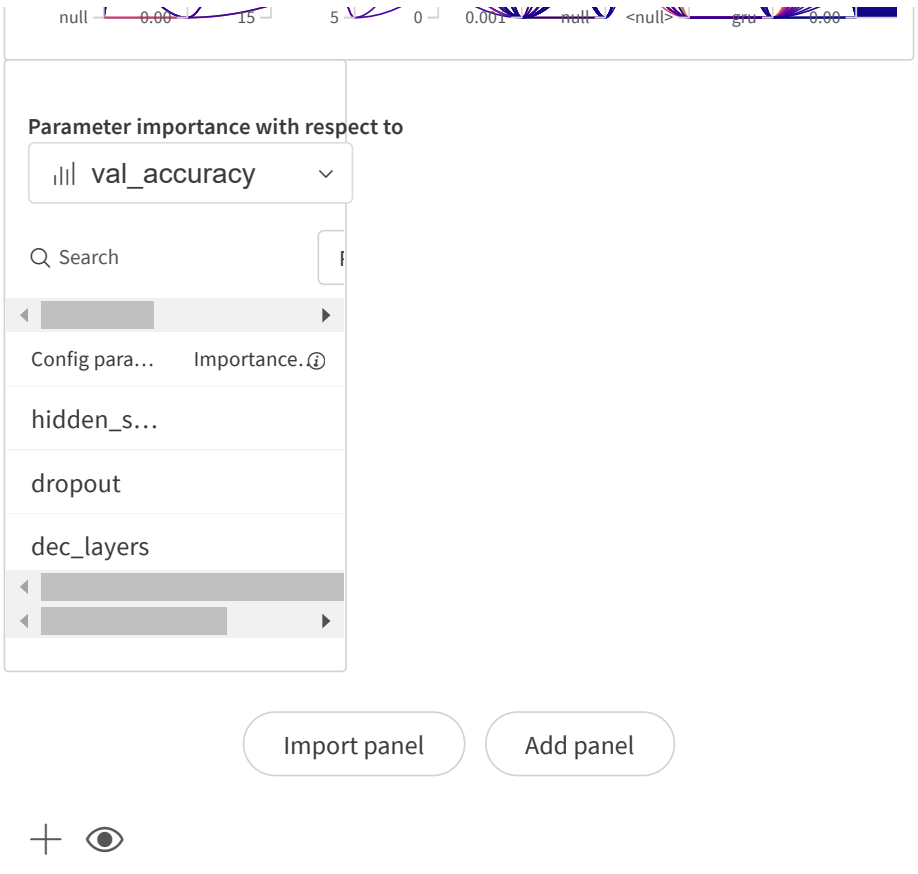




Import panel Add panel







Answer

I swept over following parameters:

- input embedding size: 16, 32, 64
- number of encoder layers: 1, 2, 3
- number of decoder layers: 1, 2, 3
- hidden layer size: 32, 64, 128, 256, 512
- learning rate: 0.001, 0.005, 0.01
- cell type: RNN, GRU, LSTM
- dropout: 0, 0.2, 0.3
- batch size: 1, 16, 64, 128
- Optimiser: SGD, Adam

Strategies to decrease number of runs:

I initially did a few `random` runs to get a rough idea on which parameters are not good.

- `RNN` was performing significantly poorer than `GRU` and `LSTM`
- Learning rate of `0.01` was too high and model was not able to converge
- Low hidden layer size such as `32` and `64` were not performing as good as higher values
- Higher batch sizes were significantly faster but the loss was not decreasing after a point
- `Adam` and `SGD` both gave similar performance with SGD giving slightly better, so I decided to go with just SGD

After analysing the `random` sweeps, I ran `bayes` sweep with following parameters:

- input embedding size: $16, 32$
- number of encoder layers: $1, 2, 3$
- number of decoder layers: $1, 2, 3$
- hidden layer size: $256, 512$
- learning rate: $0.001, 0.005$
- cell type: GRU, LSTM
- dropout: $0, 0.2, 0.3$
- batch size: 1
- Optimiser: SGD

Note that it was very hard to run sweeps with more count as my laptop does not have GPU and thus it is very slow and kaggle, colab have runtime limits.

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

- Cell type:
  - `GRU` and `LSTM` give way better accuracies than `RNN`
  - This can be explained by the fact that `LSTM` and `GRU` are better models for translation than plain `RNN` as they can get more information about context through gates
- Embedding size:
  - Higher values of and above perform poorer compared to lower values like `16` and `32`
  - Total number of distinct characters in the languages are around 64, so it can be easily represented by smaller embedding size. Larger values may even overfit
- Hidden layer size:
  - Smaller hidden layer size do not give good accuracies
  - 4 out of top 5 runs have hidden size `512`
  - I am not exactly sure why this is happening but it is possible that the problem is complex enough that higher hidden size is better model for it
- Learning rate:
  - Top 7 out of 10 runs have learning rate `0.005`
  - There is not much difference in lower accuracies for `0.001` and `0.005` but it is evident that more model with `0.005` accuracies were able to get more than `0.3` validation accuracy
- Dropout:
  - `0.2` and `0.3` perform better than `0` but this maybe because I ran more runs for them
- Number of layers does not have much impact on the accuracy as we can see from the correlation. This may be because their effect is over shadowed by other parameters and you may need specific runs to see the effect of number of layers

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 maybe it's just me!
- ...

Answer

(a) My best configuration is as follows:

Parameter	Values
epochs	10
optim	SGD
learning_rate	0.005
batch_size	1
hidden_size	512
embed_size	32
enc_layers	3
dec_layers	1
type	lstm
dropout	0.3

Accuracy on test set: 33.333%

(b) Sample inputs:

input_seq	target_seq	output_seq
wipaksh	વિપક્ષ	વિપક્ષ
pirasta	પીરસતાં	પીરસ્ટ
swasthyane	સ્વાસ્થયને	સ્વાસ્થ્યને
talaavno	તળાવનો	તલાવનો
parker	પાર્કર	પાર્કર
kyaan	ક્યાં	ક્યાં
jamaadaar	જમાદાર	જમાદાર
tehri	તેહરી	તેહરી
temin	તેમીન	ટેમિન
parkmaan	પાર્કમાં	પર્કમાં
phutabolar	ફૂટબોલર	ફુટાબોલર

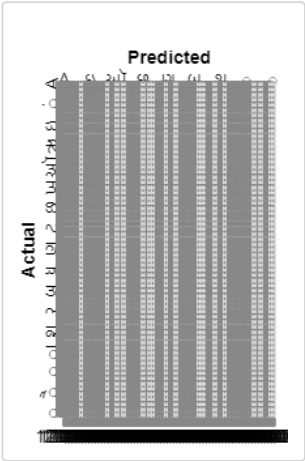
(c) Following are the common errors I have observed and we can infer from confusion matrix as well.

I am not sure how to explain these to someone if they don't understand `Gujarati` but the issue is that for almost all types of consonants and vowels, there is another which is very similar. So, the model is not able to give the output exactly but very close.

- અ <--> આ
- ઇ <--> ઈ
- ઉ <--> ઊ
- એ <--> ઐ
- ઓ <--> ઔ
- દ <--> ધ
- ન <--> ણ
- પ <--> ફ
- બ <--> ભ
- મ <--> ં
- ' <--> અ
- ત <--> ટ
- થ <--> ઠ
- દ <--> ડ
- ર <--> ઋ <--> ' ૃ
- લ <--> ળ

- ર્ા <--> ષ <--> ત

Above mentioned character can be used in similar contexts and sound very similar. So the model can't exactly decide which one is the correct at that place. If we allow interchanging of these characters, then accuracy increases by significant amount.



Import panel Add panel

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.

(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

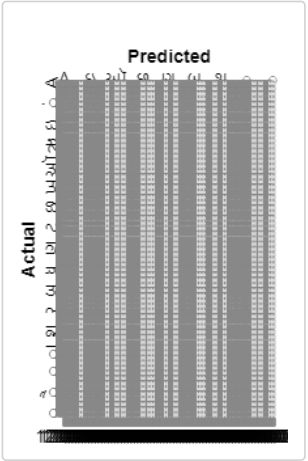
(a) I did not tune the hyperparameters again due to lack of time.

(b) My best model gave test accuracy of 34.18%

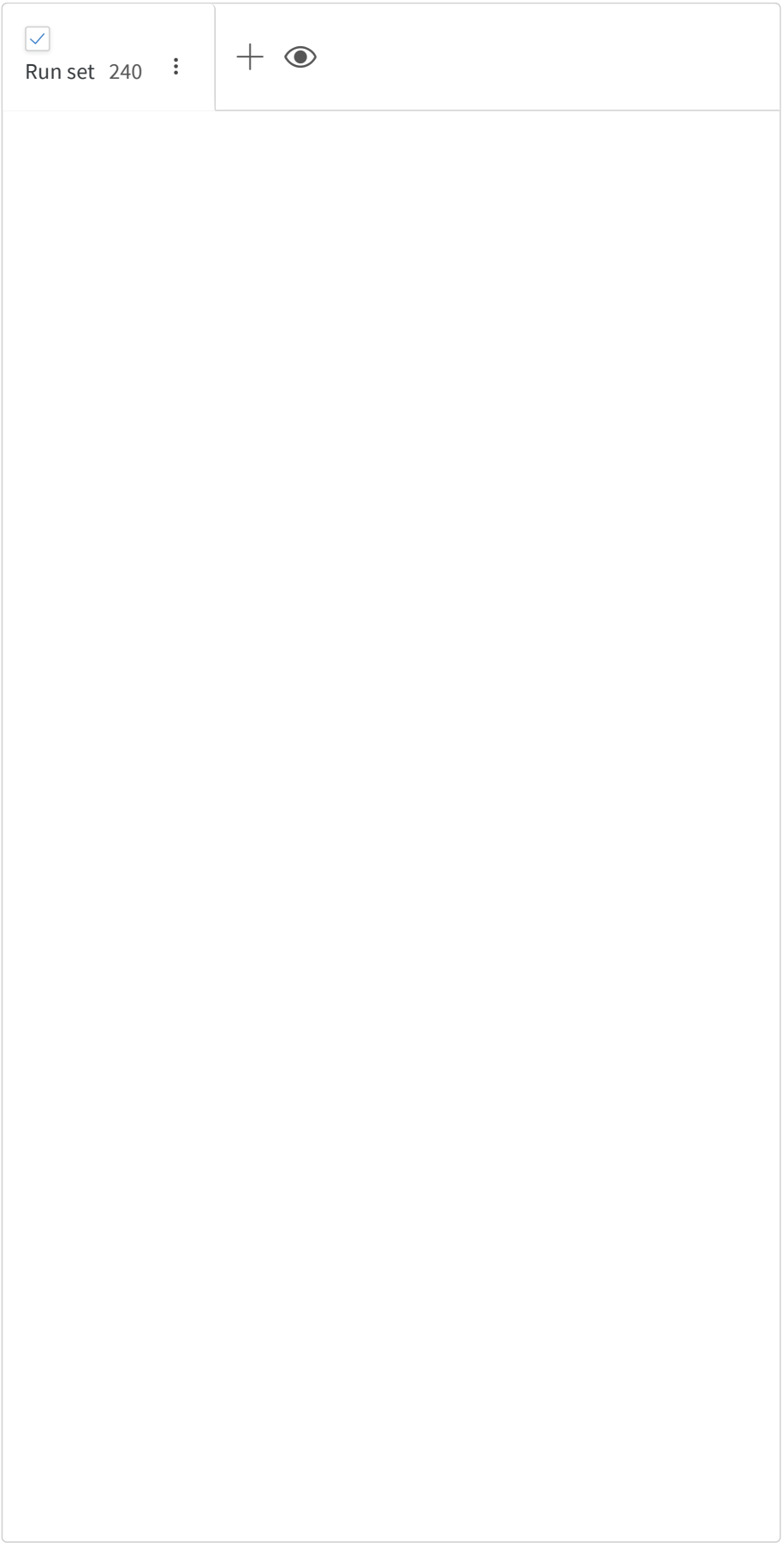
(c) The model with attention performs slightly better than vanilla model. Here are some of the errors corrected by attention:

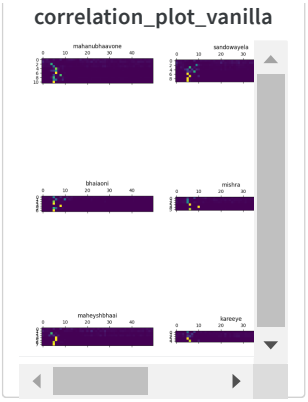
input	attention	vanilla
mahanubhaavone	મહાનુભાવોને	મહાનુભવોને
sandowayela	સંડોવાયેલા	સંદોવાયેલા
bhaiao	ભાઈઓ	ભાઇઓ
bhaiaoni	ભાઈઓની	ભાઇઓની
mishra	મિશ્રા	મિશ્ર
kabajama	કબજામાં	કબાજમાં
maheyshbhaai	મહેશભાઈ	મહેશભાઇ
kareeye	કરીયે	કરીએ
kalaapreymee	કલાપ્રેમી	કલાપ્રેમીય
patana	પટના	પાટાના
modhaaman	મોઢામાં	મોધામાં

I have pasted confusion matrix for this case as well. As we can see, some of the errors mentioned above are fixed by model with attention and it is able to predict more accurately.



Import panel Add panel


Run set 240



Import panel Add panel

(UNGRADED, OPTIONAL) Question 6 (0 Marks)

This 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

Answer: https://github.com/Mantra-7/CS6910-Assignment3

(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, Mantra (Roll no: CS20B083), 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.