ME19B168 CS6910 - Assignment 3

Use recurrent neural networks to build a transliteration system.

Shrung D N

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:

(Note: I have used the code structure from the blog given in this assignment sheet as reference and have built my code around it)

a) Taking the network to be an RNN model, the following computations are required:

- The hidden state is found by the matrix multiplication followed by additions (s is the previous state, x is the input): Vx + Ws + b
    - V is of shape k x m
    - x is of shape m x 1
    - b is of shape m x 1
    - W is of shape k x k

- The number of computations done by a matrix of matrix of shape B x A on a vector of shape A x 1 are:
  - A x B multiplications and A x B -1 additions = 2 x A x B - 1 computations
- Hence, number of computations are:
  - Vx: 2 x k x m -1 computations
  - Ws: 2 x k x k -1 computations
- This is followed by 3 x (k - 1) additions
- Hence total number of computations in encoder layer is: (2km-1) + (2kk-1) + (3k - 3) = 2k(m+k+3)-5 computations per time step
- The decoder layer layer has an additional output_func(Us + b) as compared to encoder. (U is of size T x k and b is of size T x 1)
  - Us: 2 x T x k -1 computations
  - + b: T computations
  - output_func: T computations
- Hence, total number of computations in decoder layer is: 2k(m+k+3)-5 + (2Tk-1) + (T) + (T) computations
- It can be simplified to: 2k(m+k+T-3) + 2T -6 computations
- Model architecture:
  - k = 128
  - m = 128
  - T = 63
- Therefore, total number of computations are: **81016** per time step

b) The number of parameters in the model is as follows:

- Based on the previous dimensions, the number of parameters are:
  - W: k x k parameters
  - V: k x m parameters
  - b: m x 1 parameters
  - U: T x k parameters
  - b: T x 1 parameters
- Total number of parameters are:

- k*k + k*m + m + T*k + T
- Using the above model architecture, total number of parameters is: **41023**

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.

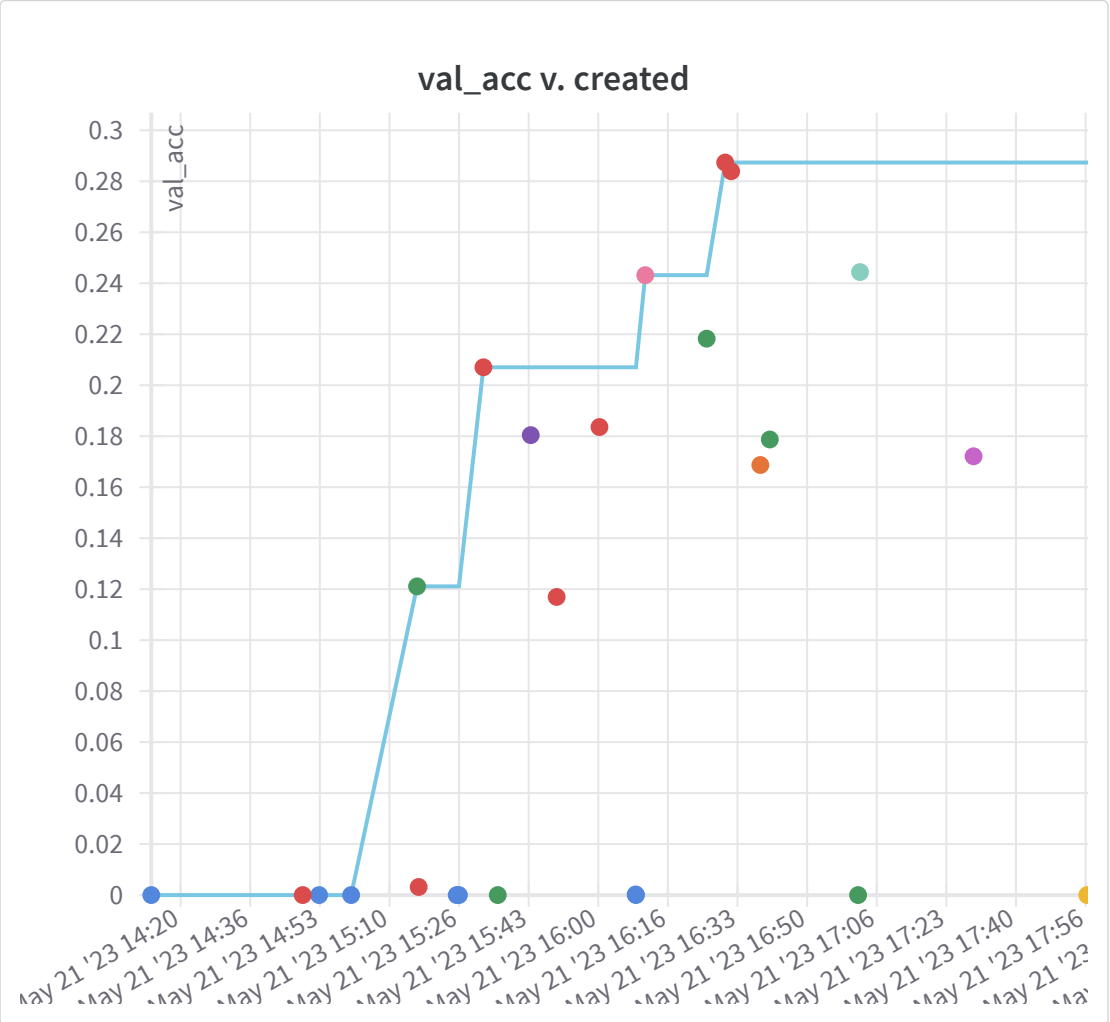
Using the sweep feature in wandb find the best hyperparameter configuration. Here are some suggestions, but you are free to decide which hyperparameters you want to explore

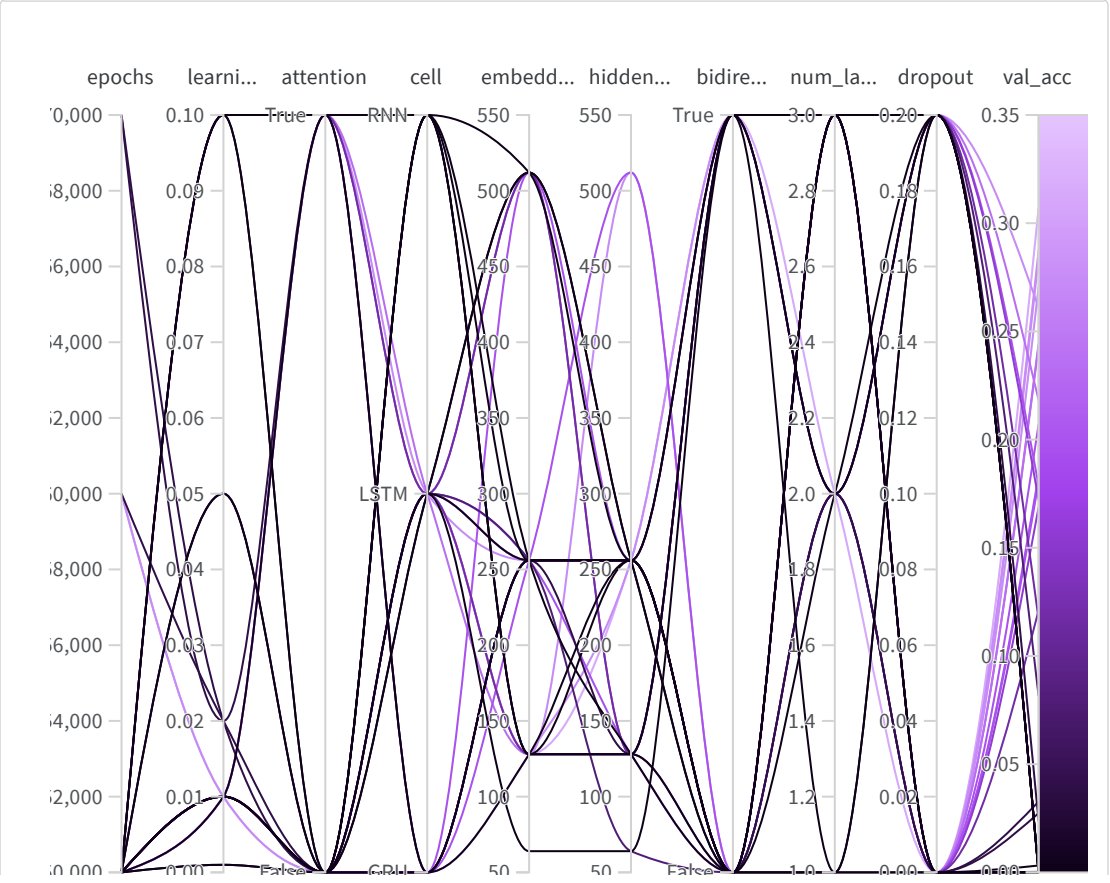
- input embedding size: $16, 32, 64, 256, \ldots$
- number of encoder layers: $1, 2, 3$
- number of decoder layers: $1, 2, 3$
- hidden layer size: $16, 32, 64, 256, \ldots$
- cell type: RNN, GRU, LSTM
- bidirectional: Yes, No
- dropout: $0.2, 0.3$ (btw, where will you add dropout? You should read up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

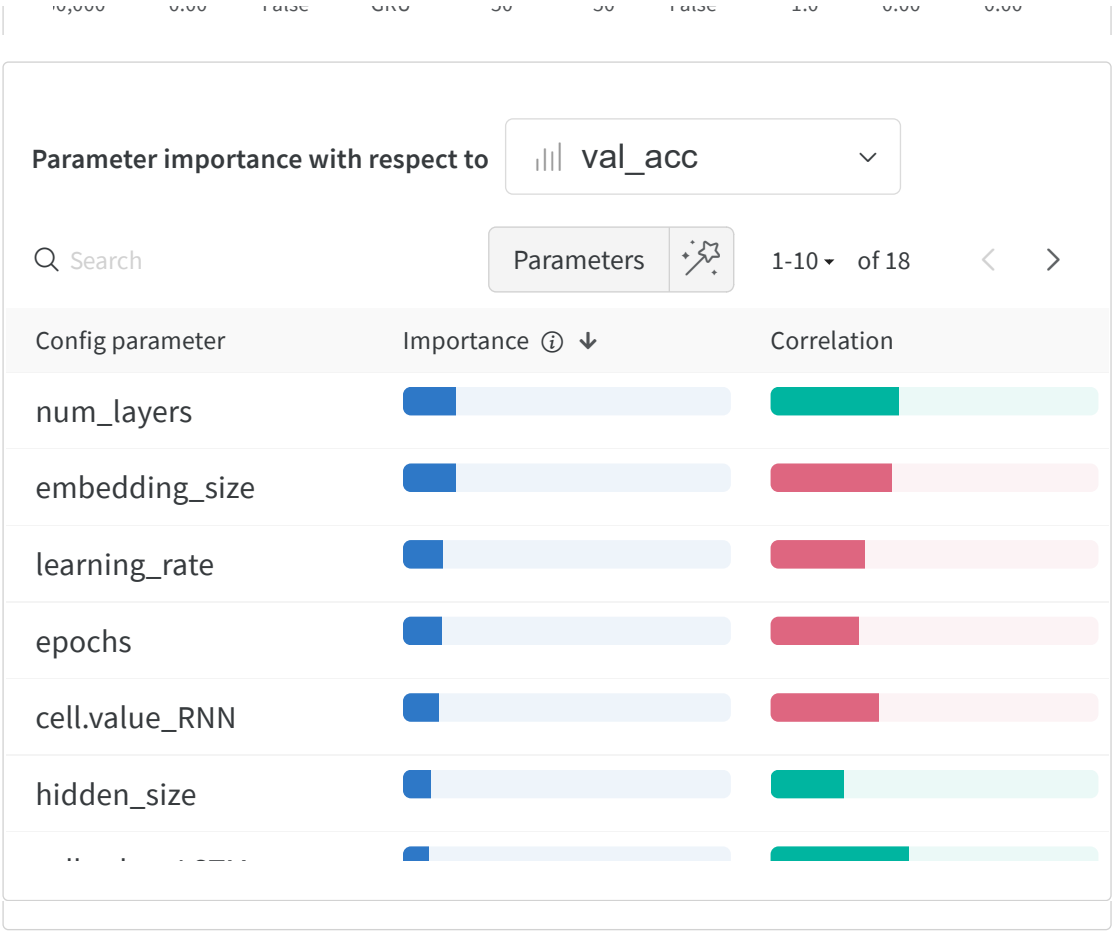
Based on your sweep, please paste the following plots, which are automatically generated by wandb:

- 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 parameters were kept constant throughout the sweeps:

- Max Length: 30(It is the maximum length of words to be used for training)
- Optimizer: SGD
- Loss: NLLLoss
- Teacher Forcing Probability: 0.5

(Note: the code is written such that the outcome of NLLLoss is similar to what would've been obtained through CrossEntropyLoss after minor edits)

The hyper parameters that I swept over are:

- Cell Type: RNN, GRU, LSTM
- Embedding Layer Size: 64, 128, 256, 512
- Hidden Layer Size: 64, 128, 256, 512
- Number of Encoder-Decoder Layers: 1, 2, 3
- Bidirectional: True, False
- Dropout: 0, 0.2
- Learning Rate: 0.1, 0.05, 0.02, 0.01, 0.001
- Epochs/Iters: 50000, 60000, 70000
- Attention: True, False

Note: Epochs/Iters refers to the number of updates done to the parameters using SGD with a batch size of 1. The reason for using SGD are:

- RNNs are sequential in nature and hence cannot be parallelized directly without padding (I had tried using padding but it gave very bad results and hence have not used it in the sweeps)
- SGD performs better in such scenarios compared to algorithms like Adam because of the high variance in updates. Algorithms that rely on momentum will perform poorly due to this high variance.

The best hyper parameters that were found for the model (without attention) is:

- Cell Type: LSTM
- Embedding Layer Size: 128
- Hidden Layer Size: 256
- Number of Encoder-Decoder Layers: 2
- Bidirectional: True, False
- Dropout: 0, 0.2
- Learning Rate: 0.01
- Epochs/Iters: 60000
- Attention: False

The strategy I followed is:

- I first began random sweeps. This gave me an overall idea of what hyper parameters performed worse and what didn't.
- I filtered out the extremely bad hyper parameters (such as using RNNs) and then began sweeps using bayes configuration to automatically filter out bad hyper parameters.
- As the time taken by the training algorithm is very high, relying completely on bayes sweep takes a lot of time, as bayes sweep requires many sample points from each hyper parameters. As this is not feasible to do efficiently, I have broken down the sweeps subsequently by filtering bad hyper parameters on my own on top of the bayes algorithm filter.

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:

Some observations are:

- Epochs:
  - Increasing number of epochs with good hyper parameters gives better results
  - Increasing number of epochs with bad hyper parameters can cause the algorithm to diverge
- Learning Rate:
  - A learning rate of 0.1 and 0.05 caused very bad runs, and almost all of them had close to 0 accuracy
  - A learning rate of 0.02 performed slightly better
  - A learning rate of 0.001 was too low and the models did not get trained well enough
  - A learning rate of 0.01 gave good results
  - This shows that the seq2seq model is very sensitive to learning rate and that it must be tuned very well
- Attention:
  - Models with attention took much longer to train than models without attention, due to the increase in number of parameters.
  - Because of the increase in number of parameters, the model need to be trained much longer with a lower learning rate to achieve good

results.
    - Hence, as the number of updates was fixed in this case, attention models performed poorly as compared to models without attention
    - However, with sufficient training epochs, attention models should perform better.
    - Attention models generally perform better with a non-zero dropout
- Cell:
    - RNNs performed very bad and most of them resulted in 0 accuracies
    - GRUs performed good. The best GRU model resulted in a validation accuracy of 27%
    - LSTMs performed the best. The best LSTM model resulted in a validation accuracy of 30%
- Embedding Size:
    - Embedding size of 64 performed poorly - due to very less dimensions
    - Embedding size of 256 and 512 performed decently - due to limited training iterations and larger number of features
    - Embedding size of 128 performed the best as it was the sweet spot between less iterations required and higher number of features
- Hidden Size:
    - A hidden size of 256 performed the most consistently
    - However, a hidden size of 128 gave the best results
- Bidirectional:
    - Enabling bidirectional enables better results due to faster training
- Num Layers:
    - A higher number of layers gave better results as it can process sequential information better without increasing the number of parameters by a lot
- Dropout:
    - Dropout in models without attention performed poorly. This is because, dropout has a cascading effect in seq2seq models due to its sequential nature. This causes loss of significant information. Hence, when used with small number of parameters, dropout performs bad.

    - Hence, dropout must be set to 0 or to a very low number.
    - Dropout could be set to a higher number in the presence of higher number of parameters, such as in the case of attention. It can also be seen from the results that dropout of 0.2 performed better with attention than without any dropout

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;

(The link to the best model and the method of evaluating it is given on github)

a) Test Accuracy of best model: 33.47%

b) Test Loss of best model: 0.914

Some sample predictions are: Syntax: >English word = True Kannada word < Predicted Word

- > kadimeyaagideyendare = ಕಡಿಮೆಯಾಗಿದೆಯೆಂದರೆ < ಕಡಿಮೆಯಾಗಿದೆಯಂದರೆ<EOS> (Correct)
- > purnawaagiruwudarinda = ಪೂರ್ಣವಾಗಿರುವುದರಿಂದ < ಪುರ್ಣವಾಗಿರುವುದರಿಂದ<EOS> (Inorrect)
- > javabdararaagiruttare = ಜವಾಬ್ಧಾರರಾಗಿರುತ್ತಾರೆ < ಜವಬ್ಧರಾಗಿರುತ್ತಾರೆ<EOS> (Incorrect)
- > hajipur = ಹಾಜಿಪುರ್ < ಹಾಜಿಪುರ<EOS> (Incorrect)
- > davadeya = ದವಡೆಯ < ದವದೆಯ<EOS> (Incorrect)

Some reasons for the above errors are:

- One of the main reasons for the incorrect predictions could be due to incorrect word representation in the english alphabet. For example, a better english word representation could be "poorna" instead of "purna", as it explains the kannada pronunciation of the word better. The same has been the case in the fourth example.
- The above error happens because of the larger number of characters in the kannada alphabet.
- In the fifth example, the "d" in the word can be interpreted in two ways. This has caused the model to predict a "dha" instead of a "d".

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:

- Best Test Accuracy with attention: 24.23%

a) The plots with and without attention have been grouped seperately in previous section.

b) The attention model performed worse than the vanilla model due to the following reasons:

- Models with attention took much longer to train than models without attention, due to the increase in number of parameters.
- Because of the increase in number of parameters, the model need to be trained much longer with a lower learning rate to achieve good results.
- Hence, as the number of updates was fixed in this case, attention models performed poorly as compared to models without attention
- However, with sufficient training epochs, attention models should perform better.

Some observations of the attentions model are:

- > dutta = ದತ್ತಾ < ಡುತ್ತಾ<EOS> (Incorrect)
- > kaanooninalli = ಕಾನೂನಿನಲ್ಲಿ < ಕಾನೂನಿನಲ್ಲಿ<EOS> (Correct)
- > sind = ಸಿಂದ್ < ಸಿಂಡ್<EOS> (Incorrect)
- > whirlpool = ವರ್ಲ್‌ಪೂಲ್ < ವಿರ್ಲಲಲ್ಪೊ<EOS> (Incorrect)
- > ruvaari = ರೂವಾರಿ < ರುವಾರಿ<EOS> (Incorrect)

Similar to the model without attention, attention models also fail in

cases where:

- similar english spellings can sound differently in different cases - this issue may be resolved when the attention network is trained for a significant amount of time
- But to obtain the result above, the attention network must have sufficient data samples with same english letters but different kannada letters and it must also be trained sufficiently.

(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

Example: https://github.com/ShrungDN/CS6910-Assignment-3

- 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.

(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, Shrung D N(Roll no: ME19B168), 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.