Assignment – 3

Name: Dipanshu Kumar Roll No: CS24M019

WANDB Link: https://wandb.ai/cs24m019-iitm/DL_Assignment_3/reports/DA6401-Assignment-3--VmlldzoxMjg2MTI3Nw?accessToken=1ikgh968g260h95lbwmq1jlm9nnpkegxnjmd2pithkxjvp4b30xjnhs12zxecwxs

Github Link: https://github.com/IITMDeep/da6401-Assignment3

Submission Instructions : kindly format your doc like this page (with your details and links) and then submit it.

Important Instructions: - - - - - - Students must follow the updated submission format strictly. Non-compliance will result in penalties. Submitting non-modular, unstructured, or unreadable code will cause penalties. Please upload all your .py files and the original .ipynb notebooks to GitHub, along with a detailed README file that explains the assignment structure, functionality, and instructions for running the code. Any deviation from it will cause penalties. You will be heavily penalized if we can't access your GitHub repo and W&B links. Any plagiarism will be reported and heavily penalized. Note that the deadlines are stringent for this assignment, as grades need to be finalized before the institute-mandated schedule.

DA6401 - Assignment 3

Use recurrent neural networks to build a transliteration system.

Dipanshu Kumar cs24m019

Created on May 20 | Last edited on May 21

Instructions

- The goal of this assignment is fourfold: (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) visualise the interactions between different components in a RNN based model.
- Collaborations and discussions with other groups are strictly prohibited.
- You must use Python (numpy and pandas) for your implementation.
- You can use any and all packages from keras, pytorch, tensorflow
- You can run the code in a jupyter notebook on colab 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 that 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 also need to 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 the Dakshina dataset released by Google. 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 (the way 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)

romanized string (gilhari) and produces the corresponding word in Devanagari (गीत).

As you would realise this is the problem of mapping a sequence of characters in one language to a sequence of characters in another language. 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 sequence of **characters** here).

Read these blogs to understand how to build neural sequence to sequence models: blog1, blog2

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, encoder and decoder have 1 layer each, the hidden cell state is k for both the encoder and decoder, 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, 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)

Solution:

Bias parameters for hidden layer = K, parameters in U = km, parameters in W = k^2, embedding layer parameters = V m, bias parameters for output layer = V , parameters in V = V k.

So the total number of parameters in one RNN =(Parameters in U + Embedding layer parameters + Parameters in V + Parameters in W + Bias parameters for output layer + Bias parameters for hidden layer) * n

Encoder and Decoder Parameters

Parameter	Encoder	Decoder
Matrix W	k^2	k^2
Matrix U	km	km
Embedding layer	Vm	Vm
Matrix V	-	Vk
Bias for hidden layer	k	k

-> For RNN Layer

- Total Number Of Parameters = Number of paramters in Decoder + Number of parameters in Encoder

```
= ( km + Vm + Vk + k + k^2 + V) + ( km + Vm + k + k^2 )
```

```
= ( Km + Vm + Vk + K + K^2 + V) + ( Km + Vm + K + K^2 )
```

```
= 2km + 2Vm+ Vk +2k + 2k^2 + V
```

Considering the SOS and the EOS tokens encoder have "T" timesteps where decoder will have "T + 1" timesteps. Paramaters are shared at every time step.

- So no Of computations = total no of parameters * total timestep

```
=((2*T+1)k^2+(2*T+1)k+(Vm+km)(2*T+1)+(T+1)V+(T+1)Vk) * O
```

-> For LSTM Layer

- There are four feedforward Network components in LSTM cell.
- Total Number Of Parameters = Number of paramters in Decoder + Number of parameters in Encoder

```
= ( 4km + Vm + Vk + 4k + 4k^2 + V) + ( 4km + Vm + 4k +4k^2 )
```

```
= 8km + 2Vm +  Vk +  8k + 8k^2  +  V
```

Considering the SOS and the EOS tokens encoder have "T" timesteps where decoder will have "T + 1" timesteps. Paramaters are shared at every time step.

- So no Of computations = total no of parameters * total timestep

```
= ( (2*T+1)4k^2 + (2*T+1)4k + (Vm+4*km)(2*T+1) + (T+1)V + (T+1)Vk ) * O
```

-> For GRU Layer

- There are three feedforward Network component in GRU cell.
- Total Number Of Parameters = Number of paramters in Decoder + Number of parameters in Encoder

```
=  ( 3km + Vm + Vk + 3k + 3k^2 + V) +  (3km + Vm + 3k + 3k^2 )
```

```
= 6km + 2Vm + 6k + Vk + 6k^2 + V
```

Considering the SOS and the EOS tokens encoder have "T" timesteps where decoder will have "T + 1" timesteps. Paramaters are shared at every time step.

- So no Of computations = total no of parameters * total timestep

```
= ( (2*T+1)3k^2 + (2*T+1)3k + (Vm+3km)(2*T+1) + (T + 1)V + (T + 1)Vk ) * O
```

Question 2 (10 Marks)

You will now train your model using any one language from the Dakshina dataset (I would suggest pick a language that you can read so that it is easy to analyse the errors). Use the standard train, dev, test set from the folder dakshina_dataset_v1.0/hi/lexicons/ (replace hi by the language of your choice)

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
- dropout: 20%, 30% (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)

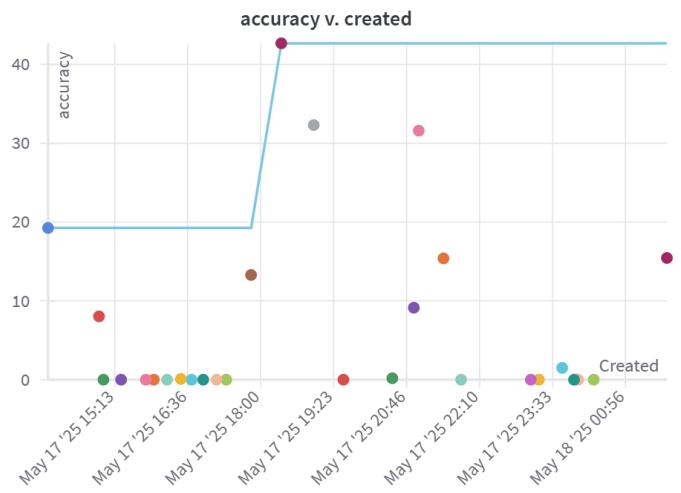
Also write down the hyperparameters and their values that you sweeped over. Smart strategies to reduce the number of runs while still achieving a high accuracy would be appreciated. Write down any unique strategy that you tried for efficiently searching the hyperparameters.

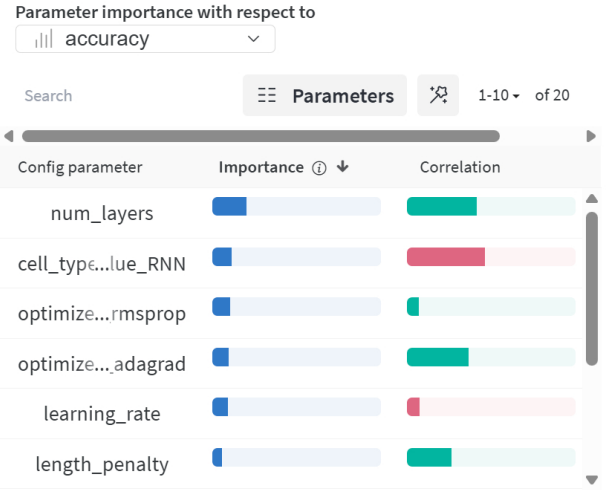
Solution:

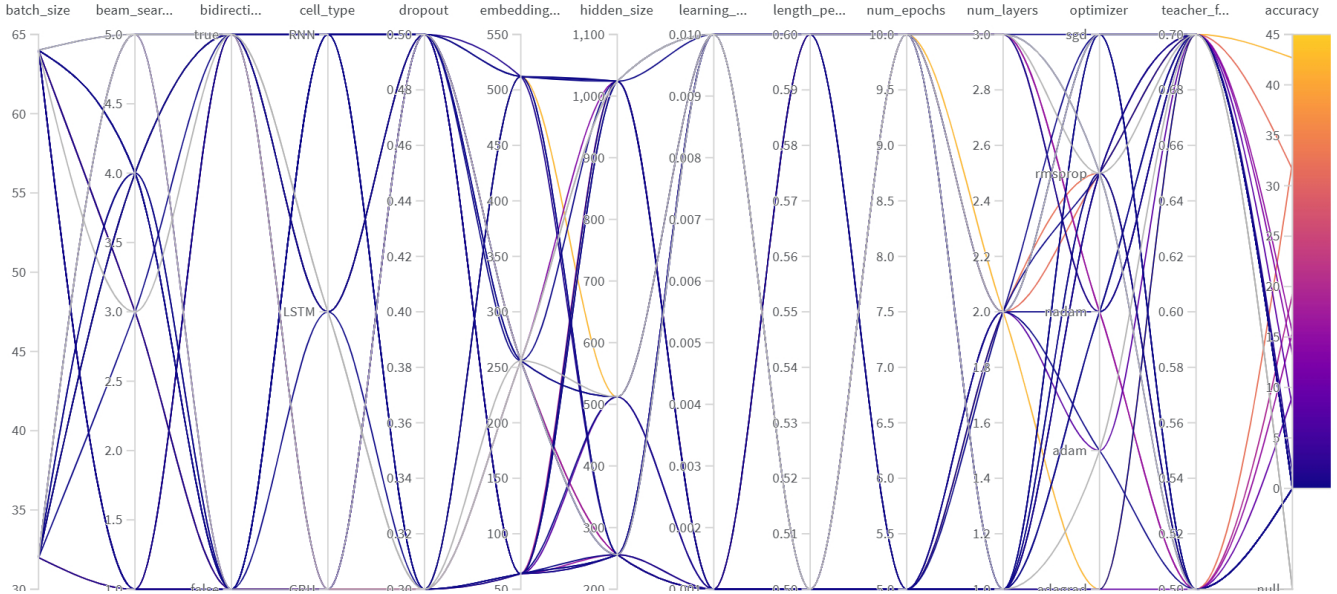
- Instead of trying every single option for the hyperparameter values, I randomly picked a variety of them. I used the **random method** in the sweep to test different hyperparameter settings. This allowed me to get a good idea of which combinations might work best, even though I didn't test every single one. This was better as we also took care of the constrained resources.
- I ran 30 sweeps using this method and then selected the best-performing configurations (based on accuracy) to use in the next **Bayesian** optimization phase.

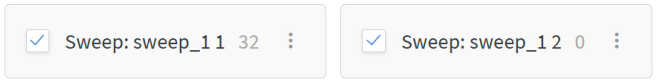
```
sweep_config = {
            'name': 'sweep : random',
            'method': 'random',
            'metric': { 'goal': 'maximize','name': 'Accuracy'},
            'parameters':
                {
                    'num_epochs': {'values': [10]},
                    'cell_type': {'values': ['RNN', 'LSTM', 'GRU']},
                    'embedding_size': {'values': [128, 256, 512, 1024]},
                    'hidden_size': {'values': [128, 256, 512, 1024]},
                    'num_layers': {'values': [1, 2, 3, 4]},
                    'dropout': {'values': [0.3, 0.5, 0.7]},
                    'optimizer' : {'values' : ['adam', 'sgd', 'rmsprop', 'adagrad']},
                    'learning_rate': {'values': [0.005, 0.001, 0.01, 0.1]},
```

```
                'batch_size': {'values': [32, 64, 128]},
                'teacher_fr' : {'values': [0.3, 0.5, 0.7]},
                'length_penalty' : {'values': [0.5, 0.6]},
                'bi_dir' : {'values': [True, False]},
                'beam_width': {'values': [1, 2, 3, 4, 5]}
            }
        }
```



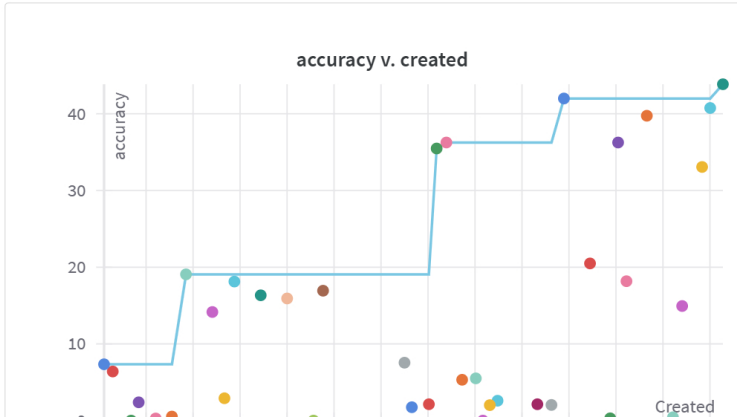


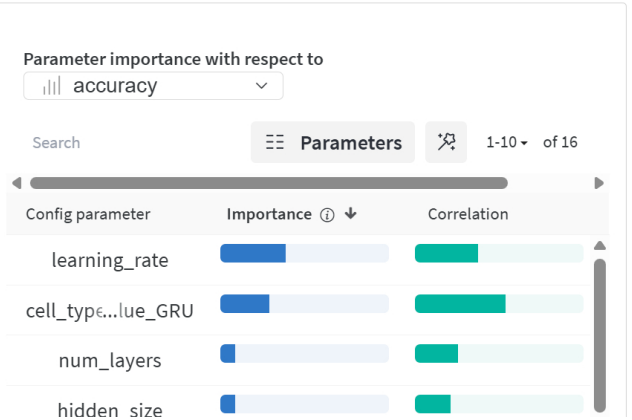


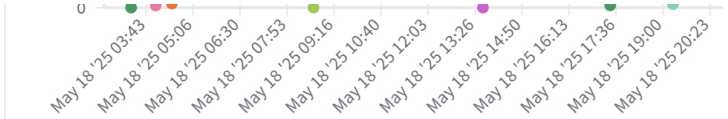


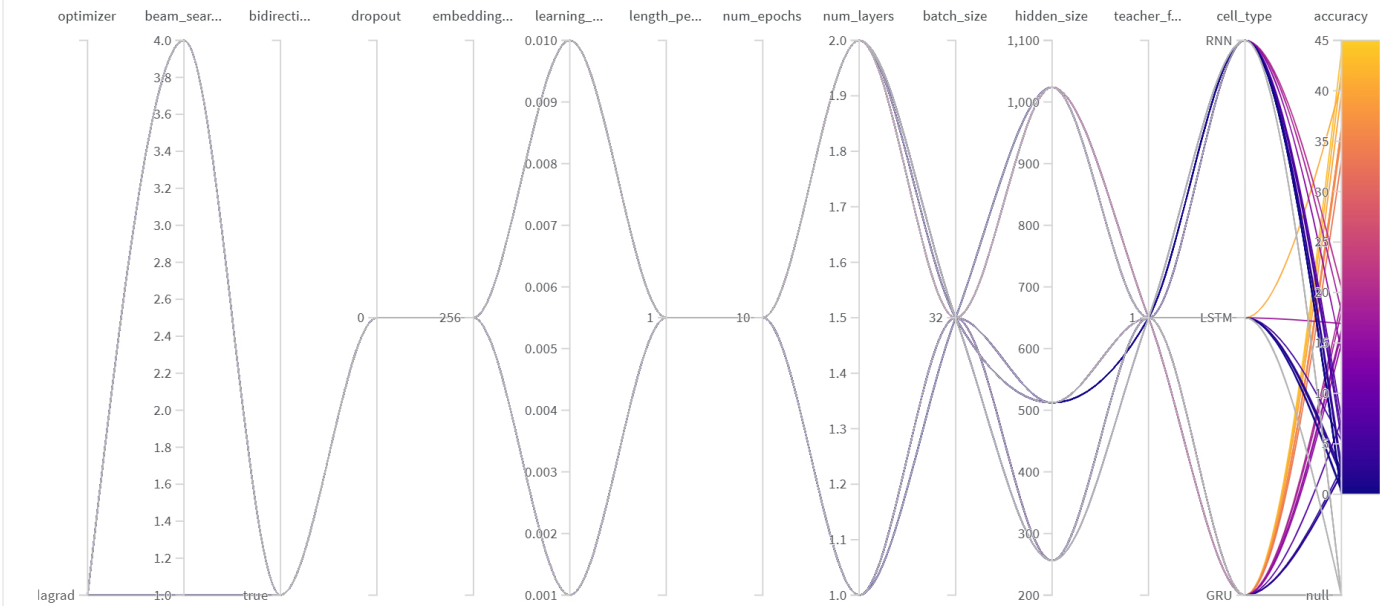
- After seeing the graphs, I got to know what are the important parameters for the accuracy. After that, I ran **45 more** experiments. I limited the range of some parameters based on the co-relation matrix. To pick the best combination, I used **Bayesian** optimization which chooses hyperparameter configurations with the goal of improving model performance with each iteration. From this sweep, I achieved the best accuracy of **43.85 %.**

```
sweep_config = {
            'name': 'sweep : bayes',
            'method': 'bayes',
            'metric': { 'goal': 'maximize','name': 'Accuracy'},
            'parameters':
                {
                    'num_epochs': {'values': [10]},
                    'cell_type': {'values': ['LSTM', 'GRU']},
                    'embedding_size': {'values': [256]},
                    'hidden_size': {'values': [256, 512]},
                    'num_layers': {'values': [2]},
                    'dropout': {'values': [0.3]},
                    'optimizer' : {'values' : ['adam', 'adagrad']},
                    'learning_rate': {'values': [0.001, 0.01]},
                    'batch_size': {'values': [32]},
                    'teacher_fr' : {'values': [0.7]},
                    'length_penalty' : {'values': [0.6]},
                    'bi_dir' : {'values': [True]},
                    'beam_width': {'values': [1, 4]}
                }
            }
```









Sweep: sweep_2 1 47

Sweep: sweep_2 2 0

Strategy I used:

To effectively explore the configuration space and optimize performance, I adopted a two-phase sweep strategy. In the first phase, I conducted multiple **random sweeps** to broadly sample various specification configurations. This helped in identifying a configuration that yielded promising initial results. In the second phase, I used this selected configuration as the base and ran more focused sweeps to fine-tune the parameters. This structured approach allowed me to narrow in on high-performing settings and significantly improved the final accuracy, especially in the **Bayes method**.

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

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

Solution:

Some Insightful Observations:

- Models that utilized bidirectional processing achieved better performance compared to models without bidirectional processing enabled.
- The relationship between the learning rate and model accuracy showed a slight negative correlation.
- Out of all the optimizers Adagrad was giving better results while Sgd performed badly.
- There was a slight negative correlation observed with embedding size and embedding size of 256 provided perfromance.
- Models accuracy increased with an increase in number of the epochs.
- batch size of 32 gave better results compared to batch size of 64.
- Model's performance showed a positive correlation with Teacher forcing and provided better outputs with at 0.5.
- In the evaluation, it was observed that character-level accuracy consistently surpassed word-level accuracy. Interestingly, certain models exhibited higher character-level accuracy compared to others, while some models achieved higher word-level accuracy despite lower character-level accuracy.
- The effect of increasing the number of encoder-decoder layers on model performance varied across different hyperparameter configurations. In certain cases, adding more layers resulted in improved performance, while in other cases, it did not lead to significant changes in performance.
- Surprisingly, reducing the beam size in beam search resulted in improved accuracy. Specifically, using a beam width of 1 yielded the best results.
- Implementing dropout regularization helped the in preventing the model from problem of overfitting
- The **RNN optimizer performed poorly**, yielding significantly lower accuracies compared to other methods, indicating that it may not be well-suited for this specific task or configuration.

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).

Solution:

Best Model Config:

```
params = {
    'embedding_size': 256,
    'hidden_size': 512,
    'num_layers_enc': 2,
    'num_layers_dec': 2,
    'cell_type': 'LSTM',
    'dropout': 0.3,
    'optimizer' : 'adagrad',
    'learning_rate': 0.01,
    'batch_size': 32,
    'num_epochs': 10,
    'teacher_fr' : 0.7,
    'length_penalty' : 0.6,
    'beam_width': 1,
    'bi_dir' : True,
}
```

- Validation Accuracy Word Level : **43.6668 %**, Correct Prediction : **1903/4358**
- Test Accuracy : **42.71435 %**, Correctly Predicted: **1923/4502**

(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.

Solution:

```
runs.summary["Sample Inputs from Test Data"]
```

Filter

Prediction

	Input	True Hindi	Predicted Hindi
0	khurpaka	खुरपका	खुर्पका
1	acapulco	अकापुल्को	आकैलोक्योक
2	besil	बेसिल	बेसलील
3	sanskaari	संस्कारी	संक्सारी
4	dhari	धारी	धरी
5	sochne	सोचने	सोचने
6	chatushkoniy	चतुष्कोणीय	चटुक्षकोणीय

1

7	punter	पंटर	पंटर
8	shrimati	श्रीमति	श्रीमति
9	vidyaaon	विद्याओं	विद्याओं

1 - 1 of 1 Export as CSV Columns... Reset table

Run: sample_inputs_Vanilla 1

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

Solution:

Observation of errors made by my model:

- The model struggles with longer words, leading to more errors due to limited memory of earlier inputs.
- Vowels are often mis-predicted, as one English vowel can correspond to multiple Hindi sounds.
- It gets confused by characters with multiple pronunciations, like 'ou' in youth or 'ch' in chaya.
- Important vowels are forgotten, especially when they affect the form of a nearby consonant (e.g., 'o' in counteron).
- Half and joined characters (common in Hindi) are difficult for the model to predict accurately (e.g., nasht, arch).
- Errors tend to occur more toward the end of the word, even though the beginning is usually correct.

Question 5 (20 Marks)

Now add an attention network to your basis 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:

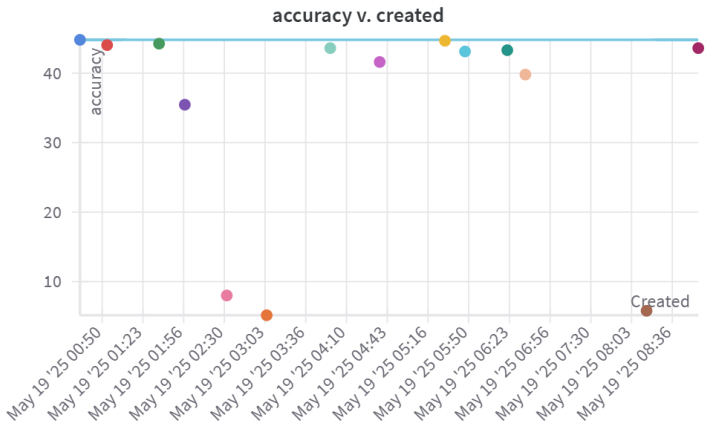
(a) Did you tune the hyperparameters again? If yes please paste appropriate plots below.

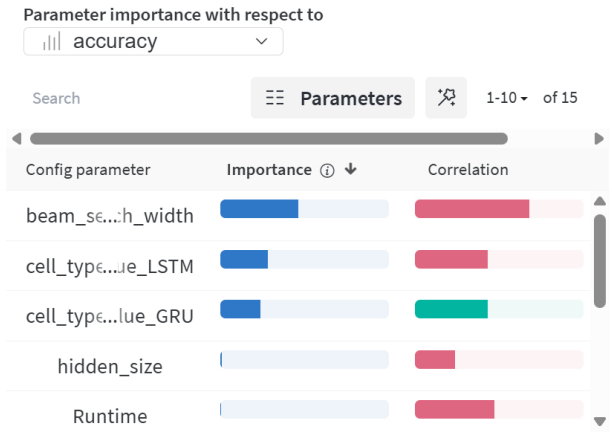
Solution:

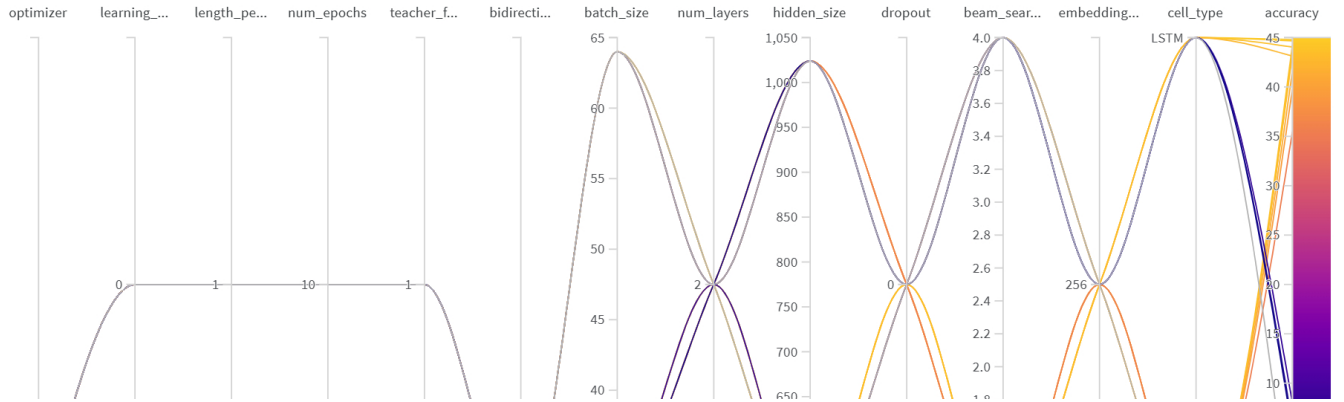
- After running Bayesian sweeps and analyzing the graphs, I understood which hyperparameters significantly influenced the accuracy. Based on those insights, I selected the best ranges for attention-based models. Since I had already narrowed down the parameter space, I used **grid search** this time to exhaustively try all possible combinations and further fine-tune the model. From this sweep, I achieved the best accuracy of **44.791 %**.

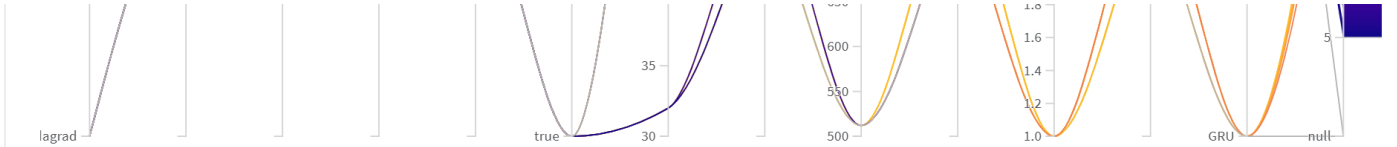
```
sweep_config = {
    'name': 'sweep_1_attention',
    'method': 'grid',
    'metric': {'name': 'accuracy', 'goal': 'maximize'},
    'parameters': {
```

```
        'embedding_size': {'values': [256]},
        'hidden_size': {'values': [512, 1024]},
        'num_layers': {'values': [2]},
        'cell_type': {'values':['LSTM', 'GRU']}, # RNN, LSTM, GRU
        'dropout': {'values': [0.3]},
        'learning_rate': {'values': [0.01]},
        'batch_size': {'values': [32, 64]},
        'num_epochs': {'values': [10]},
        'optimizer': {'values': ['adagrad']}, # ['sgd', 'rmsprop', 'adam', 'nadam']
        'beam_search_width': {'values': [1, 4]},
        'length_penalty' : {'values': [0.6]},
        'bidirectional': {'values': [True]},
        'teacher_forcing': {'values': [0.7]}
    }
}
```









Sweep: sweep_1_attention 1 16

Sweep: sweep_1_attention 2 0

(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.

Solution:

Best Model Config:

```
params = {
        "input_size": len(input_char_to_int),
        "output_size": len(output_char_to_int),
        "embedding_size": 256,
        "hidden_size": 512,
        "enc_num_layers": 2,
        "dec_num_layers": 2,
        "cell_type": "LSTM", # LSTM, GRU, RNN
        "dropout": 0.3,
        "learning_rate": 0.01,
        "batch_size": 32,
        "num_epochs": 10,
        "optimizer": 'adagrad',  # ['sgd', 'rmsprop', 'adam', 'nadam']
        "beam_search_width" : 1,
        "length_penalty" : 0.6,
        "bidirectional": True,
        "teacher_forcing":0.7,

    }
```

- Validation Accuracy Word Level : **44.7912 %**, Correctly Predicted : **1952/4358**
- Test Accuracy : **44.069 %**, Correctly Predicted: **1984/4502**

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

Solution:

Insights on Model Errors:

The primary recurring error in the model lies in predicting vowels, particularly in discerning between similar sounding vowel combinations. For instance, in words like "haashie," the model often confuses between "हाशी" and "हाशिए". Similarly, it translates "canterbury" as "केंटरब्यूरी" instead of "कैंटरबरी". This confusion also arises when two words or groups of letters sound alike.

It gave wrong result in some words like kashmir i.e, it gave 'कश्मिर' instead of कश्मीर.

The distinction between 'त' and 'ट', as well as 'थ' and 'ठ', created ambiguity when transliterating certain words from English to Hindi.

Model also got confused in what to use between 'स' and 'श'. example for svaayattshasan it gave स्वयत्तश्सन instead of स्वायत्तशासन.

The integration of attention mechanism represents a significant enhancement in model performance. This improvement is achieved by enabling the model to focus on important input words during each step of processing and to take into account the influence of neighbouring characters or elements. Despite these advancements, certain errors persist even in models that incorporate attention mechanisms like "गोंद" as "गोंड" or "लिब्रेशन" as "लिबरेशन".

-> Factors contributing to the superiority of attention models:

1. The enhanced attention mechanism enables the model to prioritize relevant input words, addressing the problem of forgetting and leading to substantial improvements in performance.
2. By effectively considering the context of surrounding characters, the attention-based model demonstrates enhanced performance, especially in scenarios where the same English alphabets have different pronunciations.

```
runs.summary["Comparison_Grid"]
```

Filter

	Word	Translation	Prediction_vanilla	Prediction_attention	Result_attention
1	haashie	हाशिए	हाशी	हाशिए	Yes
2	ambikapur	अम्बिकापुर	अंबिकापुर	अम्बिकापुर	Yes
3	sametati	समेटती	समेततो	समेटती	Yes
4	dayaalapuraa	दयालपुरा	दयालापुरा	दयालपुरा	Yes
5	saahityotsav	साहित्योत्सव	साहित्योत्तव	साहित्योत्सव	Yes
6	shikayatkarta	शिकायतकर्ता	शिकायकततो	शिकायतकर्ता	Yes
7	holt	होल्ट	हॉल्ट	होल्ट	Yes
8	vankshetra	वनक्षेत्र	वंक्षेत्र	वनक्षेत्र	Yes
9	latakakar	लटककर	लटकाकर	लटककर	Yes
10	saraaymohiuddinpur	सरायमोहिउद्दीनपुर	सरामियोहोद्दनपुर	सरायमोहिउद्दीनपुर	Yes
11	capacitor	केपेसिटर	केपासिटर	केपेसिटर	Yes
12	canterbury	कैंटरबरी	कैंटरब्यूरी	कैंटरबरी	Yes

13	kaathiyavadi	फाठियावाड़ी	फाठियावाड़ी	फाठियावाड़ी	Yes
14	herabaramee	बेरहरमी	बेरहरमी	बेरहरमी	Yes

1 - 15 of 454 Export as CSV Columns... Reset table

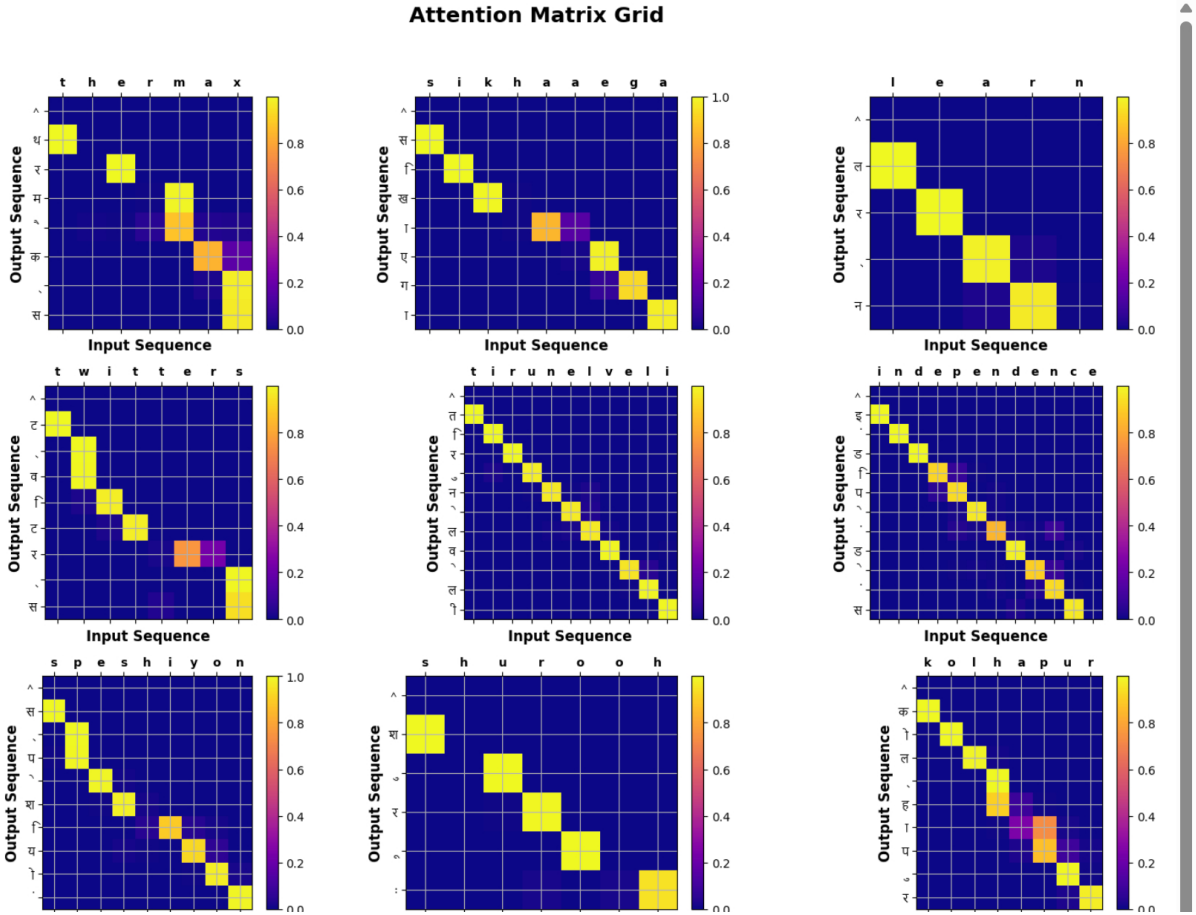
Run: Comparision 1

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

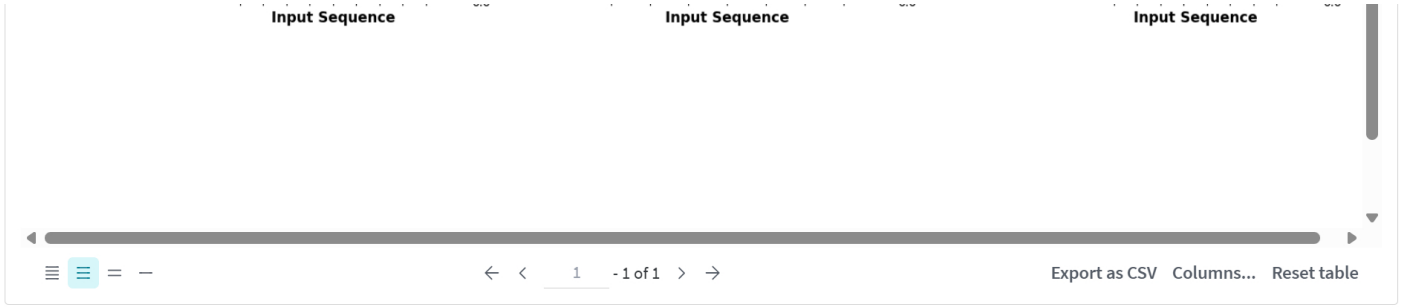
```
runs.summary["Attention Heatmap"]
```

Filter

Attention Heatmap



1



Run: heatmap 1

Question 6 (20 Marks)

This a challenge question and most of you will find it hard.

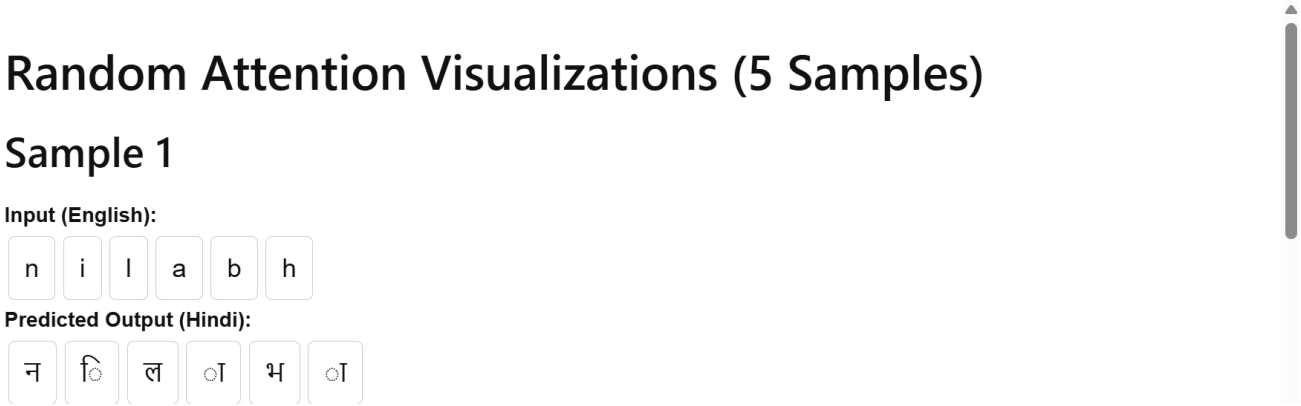
I like the visualisation in the figure captioned "Connectivity" in this article. Make a similar visualisation for your model. Please look at this blog for some starter code. The goal is to figure out the following: When the model is decoding the i-th character in the output which is the input character that it is looking at?

Have fun!

Solution:

By hovering over any **Hindi word**, you can see which **English characters** the model focuses on for generating each output character. This helps understand how the model aligns input and output during transliteration.

attention_visualizations_random_10



Sample 2

Input (English):

p e r a n o r m a l

Run: wobbly-bush-162 1

Question 7 (10 Marks)

Paste a link to your github code for Part A

Link:

https://github.com/IITMDeep/da6401-Assignment3

Self Declaration

I, Dipanshu_Kumar (Roll no: CS24M019), 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.

Add a comment