DA24M026 Assignment 3

RNNs for Transliteration

Vamsi Krishna Mohan Susarla da24m026

Created on May 16 | Last edited on May 18

Question 1

We build a Sequence-to-sequence model with an encoder (embedding included inside) encoding the given word and the decoder which outputs one character at a time. Our selections are English for the input language and Hindi as the target language. The parameters for the following sub-questions are as follows.

1. Embedding size: m
2. Hidden cell state size (encoder and decoder): k
3. Number of layers in encoder and decoder: 1
4. Length of input and output sequences: T
5. Vocabulary of input and target languages: V

Before going into the calculations, we break down the structure of the Encoder and Decoder model to simplify our work

Encoder = Embedding layer (input language) + RNN layer

Decoder = Embedding layer (target language) + RNN layer + Dense layer (to predict next character)

(a) Total number of computations (In a single forward pass)

Encoder

- For the encoder embedding layer, ideally we would multiply the onehot encoded input sequence of size $T \times V$ with the embedding matrix M of size $V \times m$ to get an embedded sequence. But multiplying a one-hot vector with 1_j with M would just yield the j^{th} column of M, and thus the matrix multiplication could just be replaced by a dictionary lookup with time complexity $O(1)$. Since we perform 1 lookup for each element in the sequence, the computation performed by the embedding layer of the encoder would be $O(T)$.
- The equations for a vanilla RNN at a given timestep t are as follows:

$$s_t = \sigma(Ux_t + Ws_{t-1} + b) \; y_t = \sigma(Vs_t + c)$$

Here, s_0 is initialised to zeros and $t = 1,2,\ldots,T$. The sizes of the given components will be as given below x_t: $m \times 1$ $s_t, b : k \times 1$

-
- $U : k \times m$
- $W : k \times k$ V
- $: m \times k$
-
- $y_t, c : m \times 1$ (Since nothing is given, we assume the sizes of x_i and y_i to be the same)
-

> Multiplying a matrix of size $m \times n$ with a vector of size $n \times 1$ is the same as performing m dot products of $n \times 1$ vectors. One such dot product takes n (multiplications) + $n - 1$ (additions) = $2n - 1$ computations thus making the total matrix-vector multiplication to have $m(2n - 1)$ computations.

The number of computations at a given timestep would be $k(2m - 1) + k(2k - 1) + k$ (addition of b) + k (sigmoid activation) for calculating s_t and $m(2k - 1) + m$ (addition of c) + m (sigmoid function) for calculating y_t.

Assumption: For simplicity, we assume the activation to be 1 computation for a single number/ element of the vector.

Total computations (timestep t) = $(k(2m-1)+k(2k-1)+2k)+(m(2k-1)+2m)$

$$= 2km + 2k^2 + 2mk + m$$

$$= 2k^2 + 4mk + m$$

For T timesteps, the total computation for the RNN layer would become $T(2k^2 + 4mk + m)$

Decoder (Character by character)

- The embedding layer in decoder will remain of the same size as that of the encoder, since the output sequence length (T) and target language vocabulary (V) are the same. Thus the number of computations will also be the same i.e. $O(1)$ for a single timestep (since the decoder works character by character) and $O(T)$ for sequence of T characters
- The RNN layer of the decoder uses the last state of the encoder RNN as it's initial state. Thus $s_0 = s_T$ (from the encoder) and the rest of the equations are as follows:

$$s_t = \sigma(Ux_t + Ws_{t-1} + b)$$

$$y_t = \sigma(Vs_t + c)$$

As the decoder is a character level RNN, to generate a sequence of length T, the decoder is run T times and at each timestep, the output at timestep t is provided as input to the decoder at timestep $t+1$. Thus, $x_{t+1} = y_t$
So even if the decoder operates differently than the encoder, it performs the same operations as the encoder. Now, since the hidden cell state for the decoder (k) and the size of input sequence of the decoder ($T \times V$) are the same as that of the encoder, the computation will also be the same.

Total computations for RNN layer = $T(2k^2 + 4mk + m)$

- The dense layer at the end of the decoder will have output size V since it predicts one character from the vocabulary at each timestep.
  The equation for the layer will be as follows:

$$y = softmax(Wx+b)$$

Here the sizes of the parameters will be: x :

- $m \times 1$ $W : V \times m$ $y,b : V \times 1$
- So, the number of computations in one forward pass of the layer will be V
- $(2m - 1) + V$ (addition of B) + V (softmax activation) thus totalling to

$2mV + V$, but for one forward pass of the model, there will be T forward passes for the dense layer amounting to
$T(2mV + V)$ computations.

Total computations (in a single forward pass)

Total computations = Encoder + Decoder

= Encoder Embedding + Encoder RNN + Decoder Embedding + Decoder RNN + Decoder Dense

$= O(T) + T(2k^2 + 4mk + m) + O(T) + T(2k^2 + 4mk + m) + T(2mV + V)$

$= T(4k^2 + 8mk + 2m + 2mV + V) + O(T)$

$= T(4k^2 + 2m(4k + V + 2) + V) + O(T)$

(b) Total parameters in the network

The total parameters in the network will be the sum of the number of parameters required in each layer of the model. There are 3 general types of layers. The parameters for those layers in general are as follows:

Embedding layer

Embedding matrix of size $V \times m$, where V = vocabulary size and m = embedding size. Thus total parameters are Vm.

RNN Layer

Multiplicative parameters U, W, b, c where the size of the parameters is as follows

- $U : k \times m$
- $W : k \times k$ V
- $: m \times k$ b :
- $k \times 1$ $c : m$
- $\times 1$

where m = input sequence vector dimension and k = hidden cell state dimension

Thus total parameters of the layer are $km + k^2 + mk + k + m = k^2 + 2km + k + m$

Dense Layer

Multiplicative parameters W and b of sizes $V \times m$ and $V \times 1$ respectively, where m and V are the input and output sizes of the dense layer respectively. Total parameters of the layer are $Vm + V$.

Total parameters of the network

Since embedding layers and RNN layers of the encoder and decoder have the same hyper-parameters and input/output shapes, the number of parameters in those layers is also the same.

Total parameters = Encoder parameters + Decoder parameters

= Encoder Embedding + Encoder RNN + Decoder Embedding + Decoder RNN + Decoder dense layer

$$= Vm + (k^2 + 2km + k + m) + Vm + (k^2 + 2km + k + m) + (Vm + V)$$

$$= 2k^2 + 4km + 2k + 2m + 3Vm + V$$

Question 2

We perform 3 sweeps in total for selecting the best model. After training a single model, it was decided to train all models for 10 epochs to give a good estimate of the metrics. The ideology of the sweeps are as follows

- First sweep is to decide the basic architecture of the network to fix upon a structure
- Second sweep takes into account, remaining parameters like dropout, embedding dimension etc. which are parameters of the network but don't fall under architecture
- Since Beam Search is just a part of inference and not training the model, and it is computationally expensive to do at each validation step, we kept a separate sweep for beam search decoder, using one of the models from previous sweeps

Teacher forcing

The encoder takes a batch of inputs and converts into a batch of encoded vectors. Since the decoder works character-by-character, it is passed <SOS> (start of sentence) tokens in the beginning along with the encoded representation to predict the 1^{st} character, then the decoder state + decoder output at timestep t is passed as input to the decoder at timestep $t + 1$.

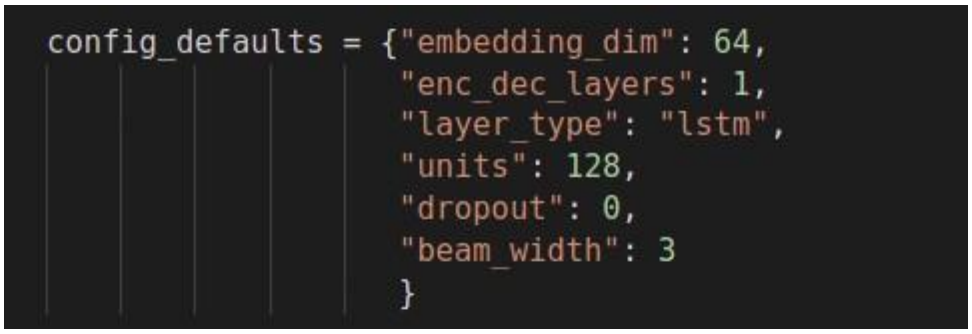
During training, we use a method called teacher forcing in which at each timestep, instead the decoder output + state as input for the next timestep, we pass the actual target values + decoder state. This makes the model learn better short term trends from the data which generalize very well.

Sweep 1

The purpose of this sweep was to fix on the architecture of the network. There were total 27 runs in this sweep and the following parameters were tuned.

- Number of layers in encoder and decoder: 1, 2, 3
- Number of neurons in a single layer: 64, 128, 256 Type of
- layer: GRU, LSTM or SimpleRNN

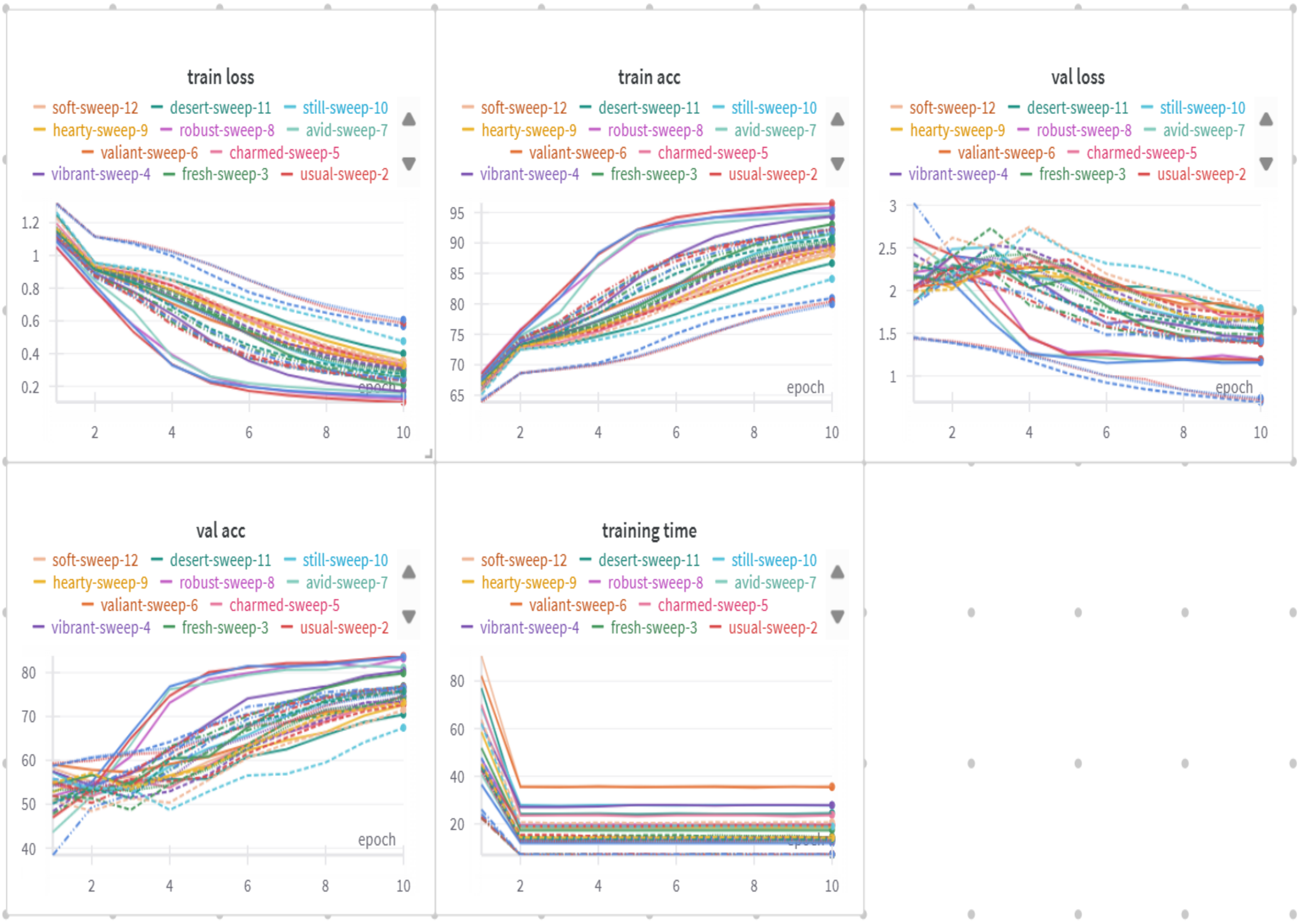
For the rest of the hyper-parameters, the following values were used as defaults



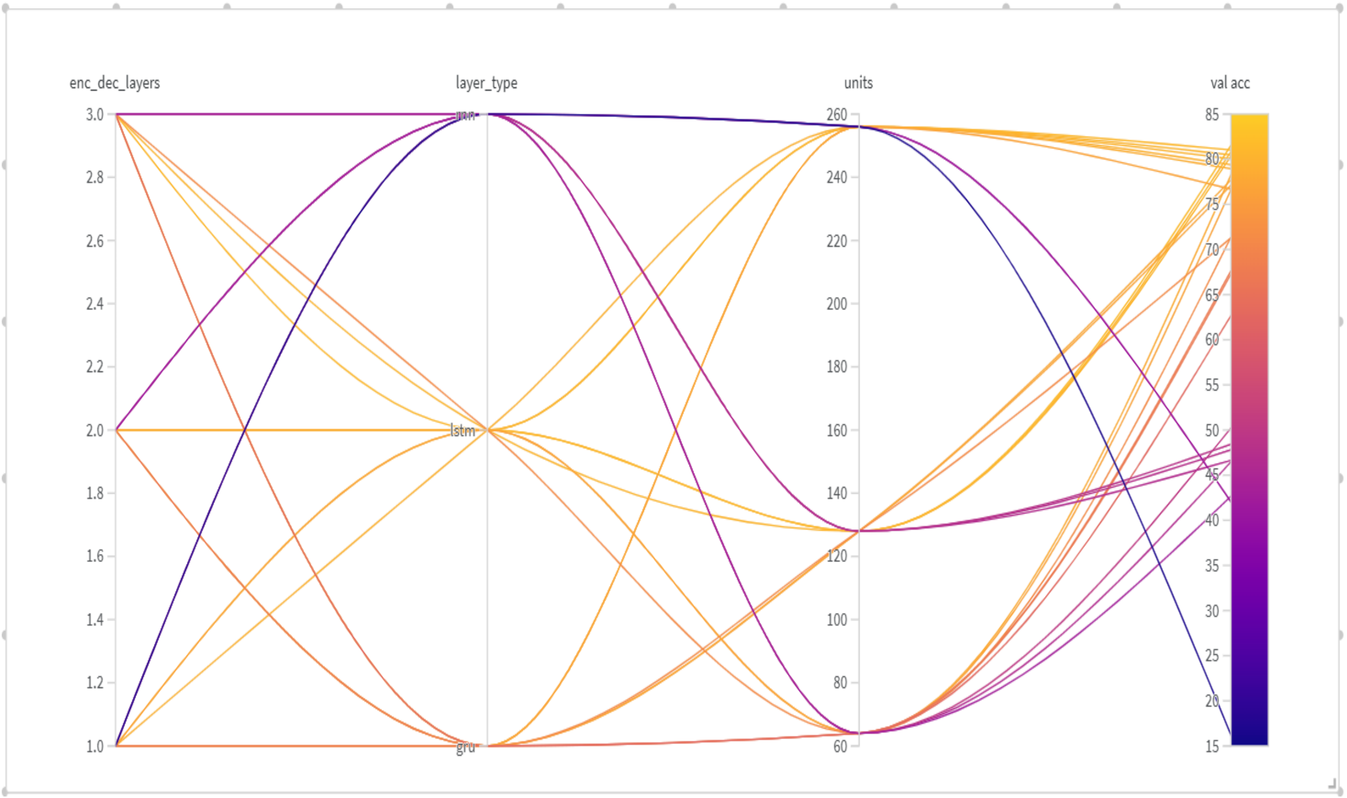
```
config_defaults = {"embedding_dim": 64,
                   "enc_dec_layers": 1,
                   "layer_type": "lstm",
                   "units": 128,
                   "dropout": 0,
                   "beam_width": 3
                   }
```

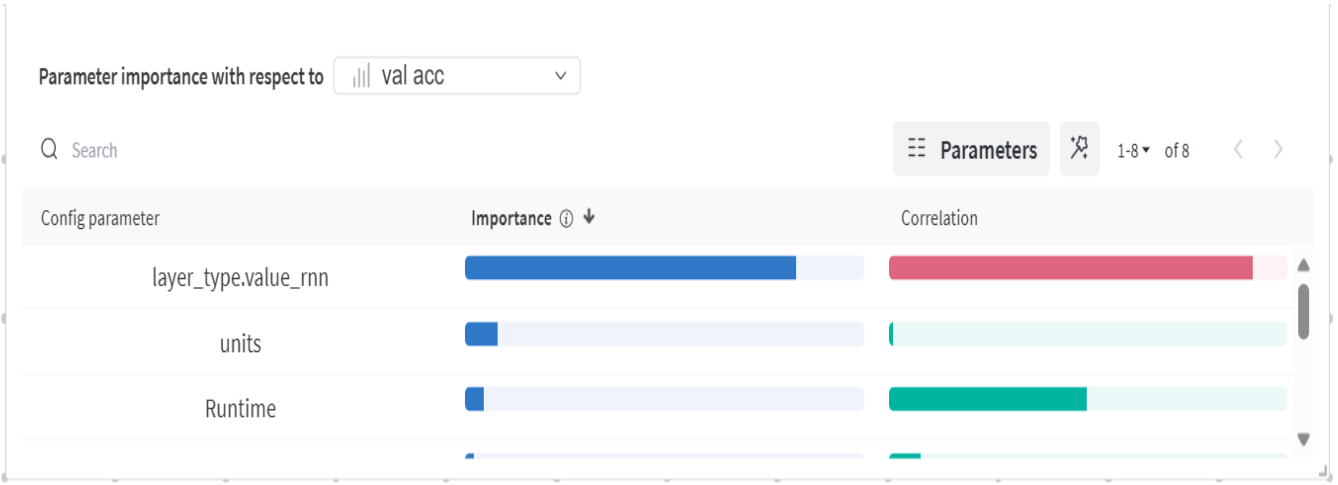
Default values of hyper-parameters

The results of the sweeps are given below,



The parallel coordinates plot of the same is given below along with importance of different parameters based on validation accuracy.




Parameter importance with respect to val acc

Config parameter	Importance	Correlation
layer_type.value_rnn		
units		
Runtime		

Inferences from Sweep 1

- LSTM works the best followed closely by GRU, but learning curve of vanilla RNN shows very erratic behaviour
- The best performance is gotten from the model with 3 layers and 256 units in each layer

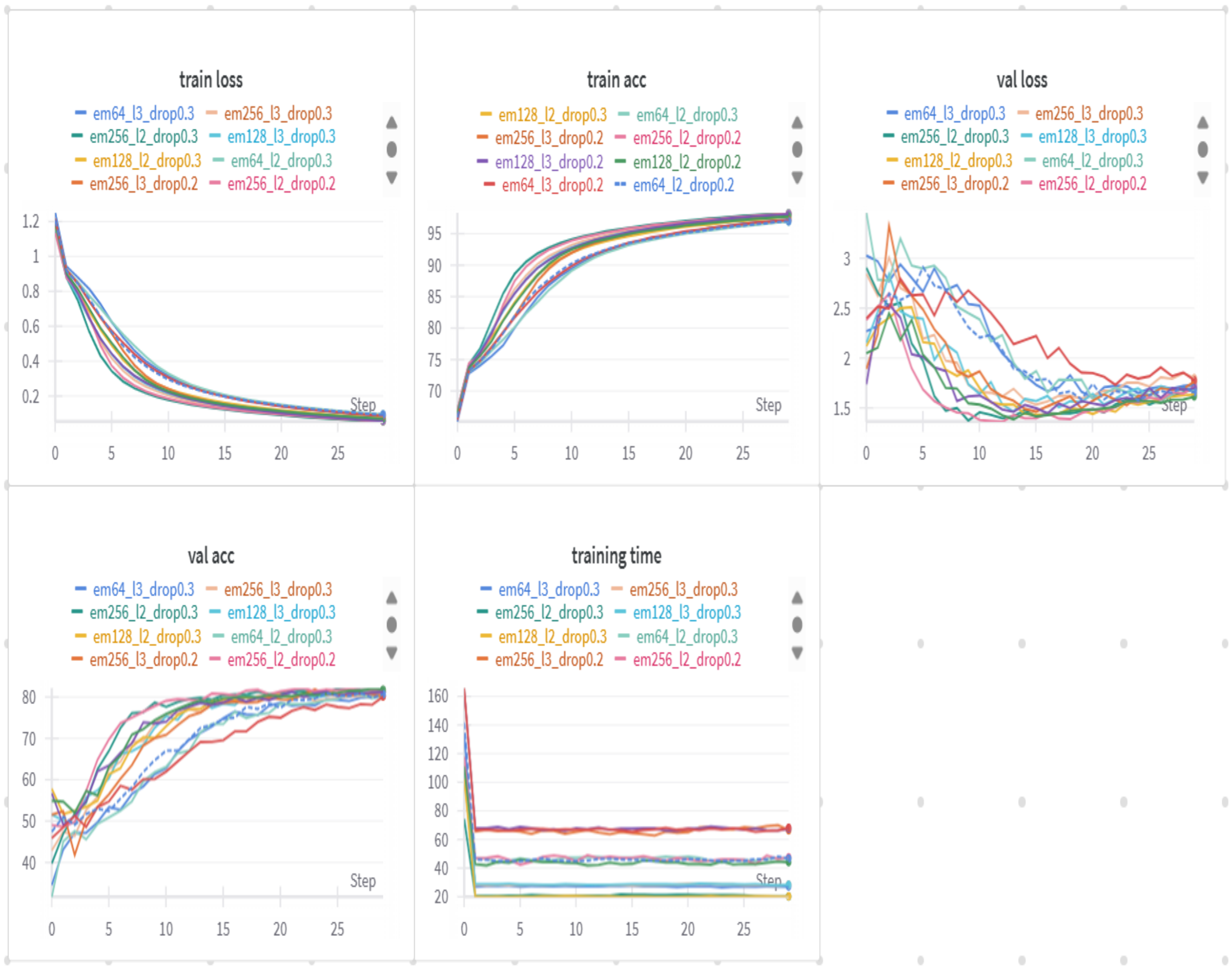
Sweep 2

In this sweep, we try to tune other parameters of the model and a total of 12 models were trained. The following parameters were used for tuning

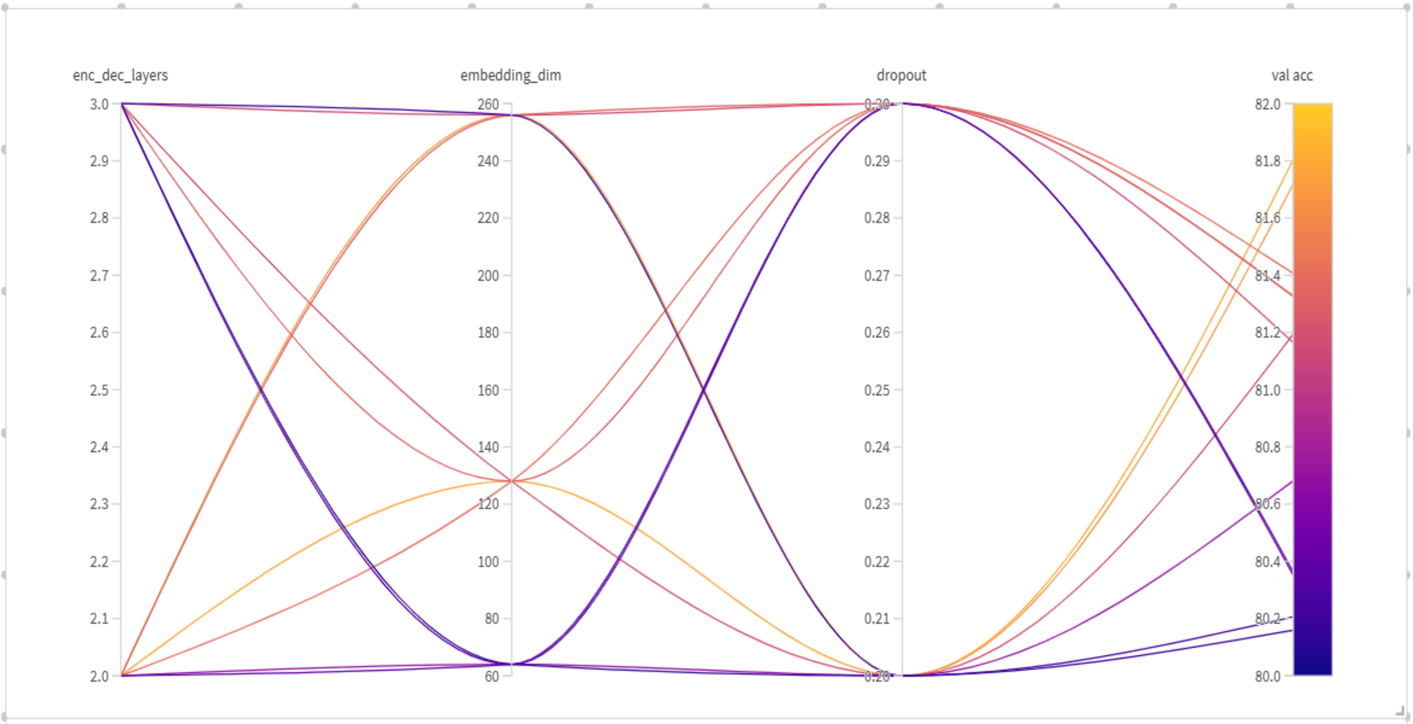
- Embedding dimension: 64, 128, 256
- Dropout: 0.2, 0.3
- Number of layers in encoder and decoder: 2, 3

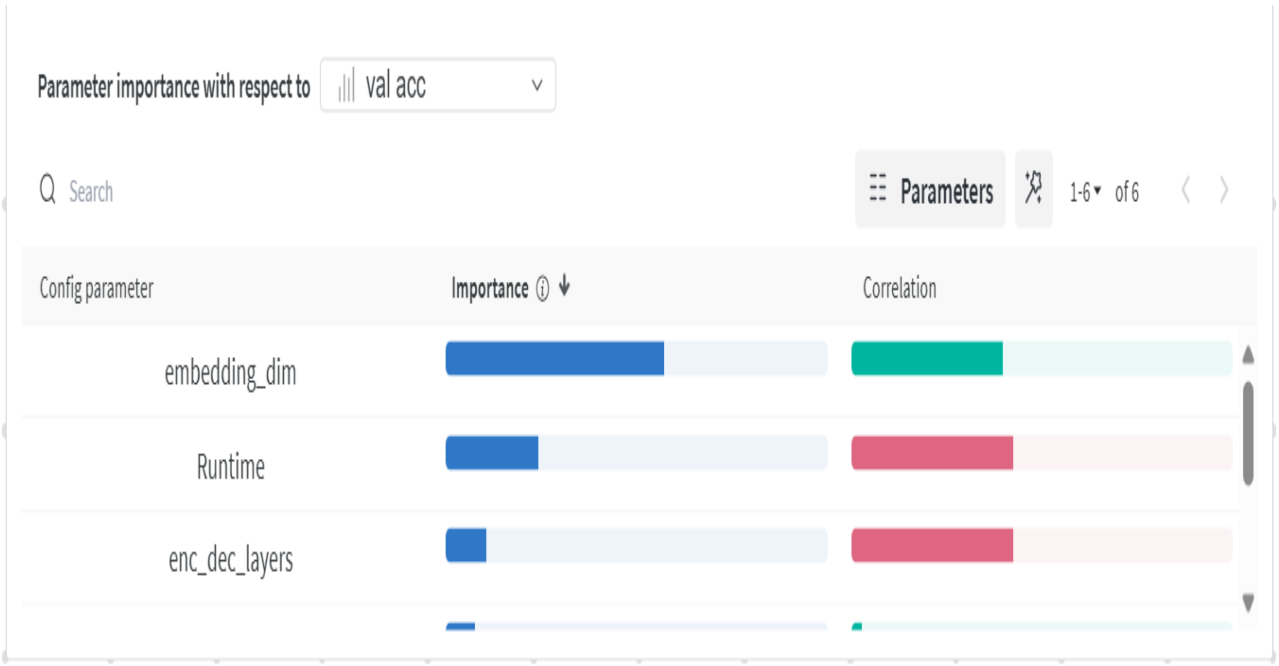
We use models with 2 and 3 layers in the encoder and decoder for this sweep and not models with a single layer because the dropout parameter provides regularisation preventing overfitting and this wouldn't be the issue in smaller models, only the bigger ones. The rest of the hyper-parameters use the default values as shown in Sweep 1.

The results of this sweep are given below.



The parameter importance table and parallel coordinates plot is give below.




Parameter importance with respect to val acc

Config parameter	Importance	Correlation
embedding_dim		
Runtime		
enc_dec_layers		

Inferences from Sweep 2

- Embedding size 128 along with a dropout of 0.2 work the best on an average based on Validation Accuracy

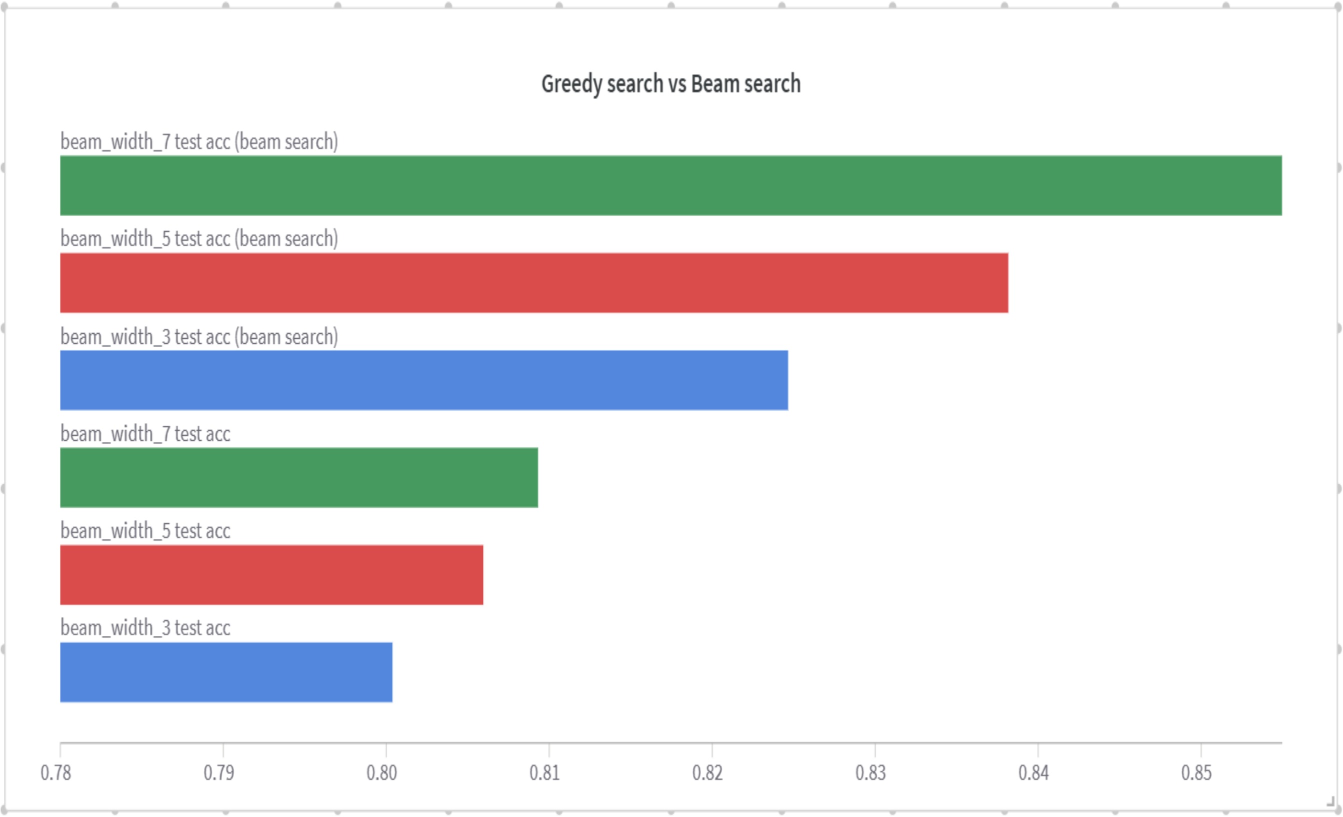
Sweep 3

This sweep was only to test the effect on beam search on existing models and consists of 3 runs. Since beam search takes a lot of time and it would be computationally expensive and also time consuming for performing sweeps, it was decided to train and validate the model in the usual way, and after the training was completed, the model would be validated on 1st 500 samples of the validation dataset itself, using both normal greedy search and beam search. The sample size was limited again to 500 samples because of the computation cost.

The following parameters were tuned and rest of the parameters were set to defaults as shown in the screenshot in Sweep 1.

beam_width: 3, 5, 7

Following are the results from the sweep. Since there's only one parameter being tuned, the parallel coordinates plot and parameter importance table are not specified.



The top 3 bars represent the validation accuracy using beam search while the bottom 3 bars are the accuracies of the same model using greedy search for respective beam sizes.

For beam search, the accuracies are calculated as follows:

- Calculate *b* (beam width) best possible sequences, using the trained model and beam search for each sample
- Calculate the character-level accuracy between *b* sequences and the ground truth and consider the maximum accuracy
- Take average of all character-level accuracies of the samples to obtain overall accuracy for the dataset

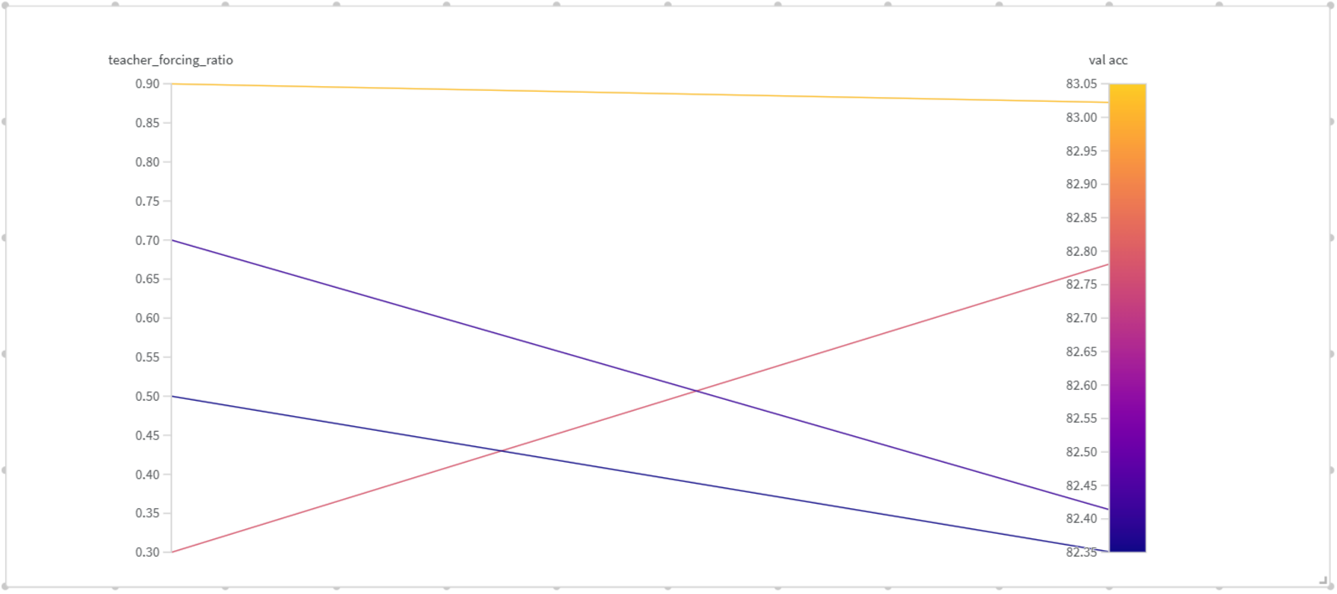
As can be seen from the above plot, more the beam size, the higher performance boost the model gets.

Sweep 4

This sweep was to test how teacher forcing affects training and how much of it should be performed while training in our case. We create a hyper-parameter named `teacher forcing ratio`. It roughly represents the percentage of teacher forcing applied to the model while in training. For example, if teacher forcing ratio is 0.7, then for each batch, there's a 70 percent chance that teacher forcing will be used to train using that batch. Following are the parameters used in this sweep.

- Teacher forcing ratio: 0.3, 0.5, 0.7, 0.9
- Number of layers in encoder and decoder: 2
- Embedding size: 128 Dropout:
- 0.2

Following are the results for the sweep



Question 3

- From Sweep 1, it's evident that LSTM based models perform the best out of all, followed closely by GRU and then Vanilla RNNs
- In our experiments, some of the validation curves for vanilla RNNs were noisy with sudden increase/decreases even if the training curves were smooth. This implies that there is a possibility that learning well on the training data doesn't guarantee good generalisation for the particular dataset
- The number of units/neurons per layer make much more difference in the validation scores as compared to the number of layers in the model, suggesting that the breadth of the model is much more important to learn on the data than it's depth
- According to our expectations, a higher number of layers should make the model learn better on the dataset, but in our experiments, we observe that they all perform pretty much similarly with models with 2 layers performing better on an average than models with 3 layers. This may be due to redundant parameters causing overfitting for higher number of layers.
- Based on the 2nd sweep, it is observed that the models with more than 1 layers perform slightly better with dropout and increased embedding size as more information is retained from the data along with dropout causing regularisation. Still, models with 2 layers in encoder/decoder outperform models with 3 layers

- In the 2nd sweep, embedding sizes 128 and 256 perform the best, with 128 even leading 256 on an average suggesting that 128 might be near the optimum encoding dimension for the data and setting it to 256 may cause redundancy making it slightly worse.
- From sweep 2, we can infer that the training time is not affected most by parameters like number of layers and embedding dimension by dropout. Higher dropout means that more outputs will be dropped (set to zero) leading to lesser computations, hence saving more training time
- While using beam search, higher beam width means that more possible sequences will be computed leading to better performance overall as it will be more likely that the ground truth is to be covered by one of the possibilities. The same can also be inferred by results of Sweep 3.
- More teacher forcing can usually result in models performing poorer since it becomes more difficult for them to predict longer sequences correctly and creates an "Exposure Bias" where the model doesn't see the inputs like what it was trained upon. But in our case since there is no requirement of long term memory, high values of teacher forcing like 0.9, 1 do work very well.

Question 4

(a) Evaluation of best model on test dataset

Based on the trends we have recognised through our sweeps, the following was the configuration set for our best model

- Embedding dimensions: 200
- Number of layers in encoder: 2
- Number of layers in decoder: 2
- Type of layer: LSTM
- Number of neurons in each layer: 500 Dropout: 0.2
-

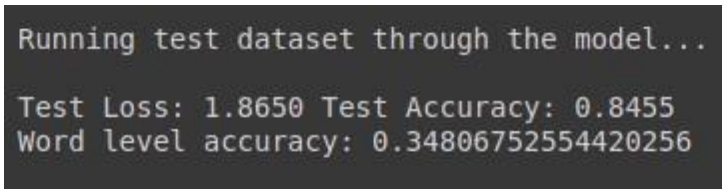
Following was the reasoning for selecting values for the hyperparameters

- Since 128 performed better than 256 for embedding size, it was concluded that increasing embedding size to 256 introduced redundant information and thus to reduce this but still keep maximum information, the embedding size is set to 200

- Number of neurons had a very strong positive correlation with the validation loss as compared to number of layers that's why it was decided to drastically up the number of neurons and take the default best value for number of layers from the sweeps
- Rest of the hyper-parameters were just the best values picked from the sweeps

Note: We did not use beam search here since we have already demonstrated it's capabilities before and running it again would be time consuming (even though it guarantees a performance boost)

The model was evaluated on the test dataset after training for 50 epochs and the following results were obtained.



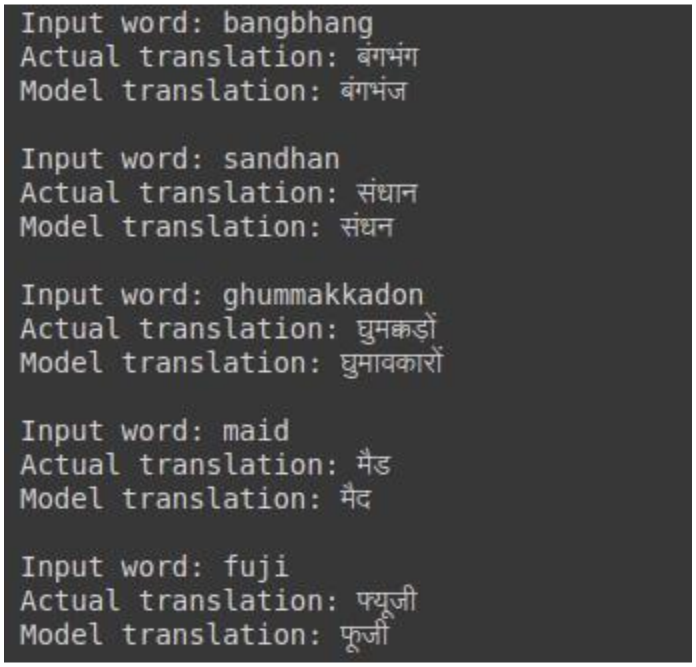
```
Running test dataset through the model...

Test Loss: 1.8650 Test Accuracy: 0.8455
Word level accuracy: 0.34806752554420256
```

Character and Word level accuracies on test dataset

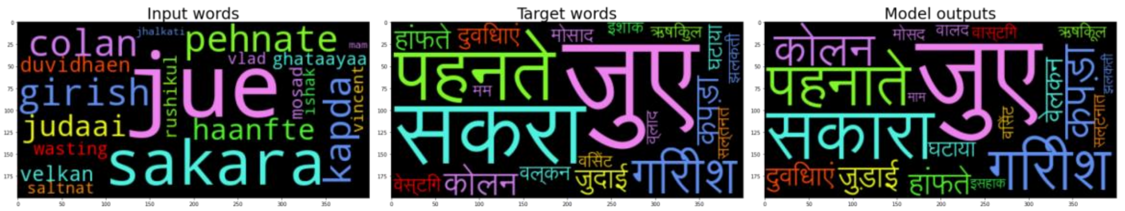
"Test Accuracy" stands for character level accuracy in the screenshot while word level accuracy is calculated such that it's 100 percent only if the entire predicted word is same as ground truth else 0 percent.

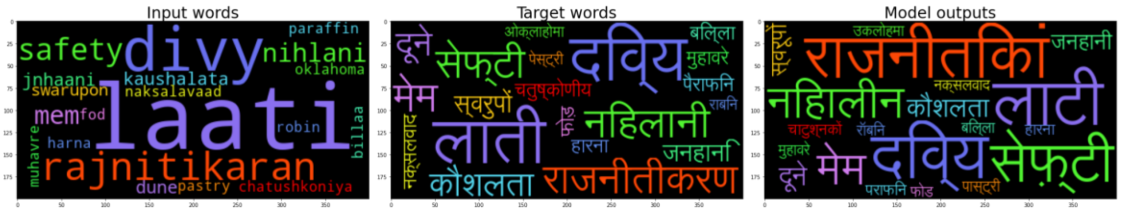
(b) Sample predictions

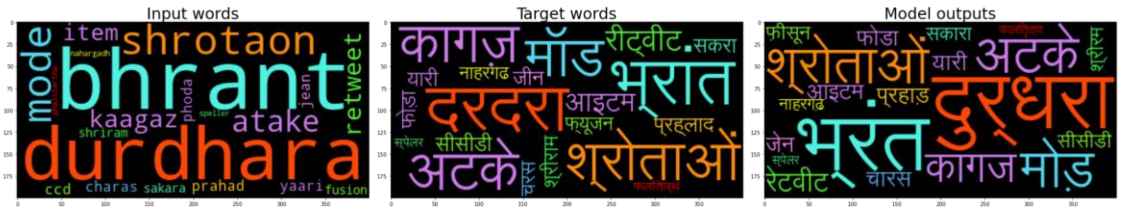
Following are some sample predictions made on the model on randomly selected samples of the test dataset


```
Input word: bangbhang
Actual translation: बंगभंग
Model translation: बंगभंज

Input word: sandhan
Actual translation: संधान
Model translation: संधन

Input word: ghummakkadon
Actual translation: घुमक्कड़ों
Model translation: घुमावकारों

Input word: maid
Actual translation: मैड
Model translation: मैद

Input word: fuji
Actual translation: फ्यूजी
Model translation: फूजी
```

We also visualise a few samples using the wordcloud python library. The colours follow a VIBGYOR spectrum and the words on the Red end are the ones which are translated poorly, and the ones translated better are towards the Violet end of the spectrum. The quality of transliteration is calculated using a smoothed BLEU score. Each triplet of input word, target word and model output are in the same colour across the images. We visualize 20 words at time in every wordcloud shown below.







(c) Comments on the model outputs

- The model makes more mistakes on longer sequences since it's difficult to catch longer term trends comparatively
- Since one vowel in English can have multiple sounds in Hindi, the model is more prone to errors on vowels rather than consonants
- The model is fairly accurate on the beginning parts of the word and as it gets to the end, the performs drops
- Since Hindi language has a lot of half/joined syllables, the model performs comparatively worse on words that contain such syllables

Question 5

a) Hyperparameter Sweep

Bahdanau Attention was implemented in the above framework, which could be used with multiple layers. Some of the hyperparameters from the previous sweep were directly used, while some hyperparameters were again tested in another sweep, this time using encoder-decoder models with attention. Hyperparameter values retained from previous sweep:

- Layer Type: LSTM
- Embedding Dimension: 256 Epochs: 30
- Hyperparameter values fixed for this sweep:

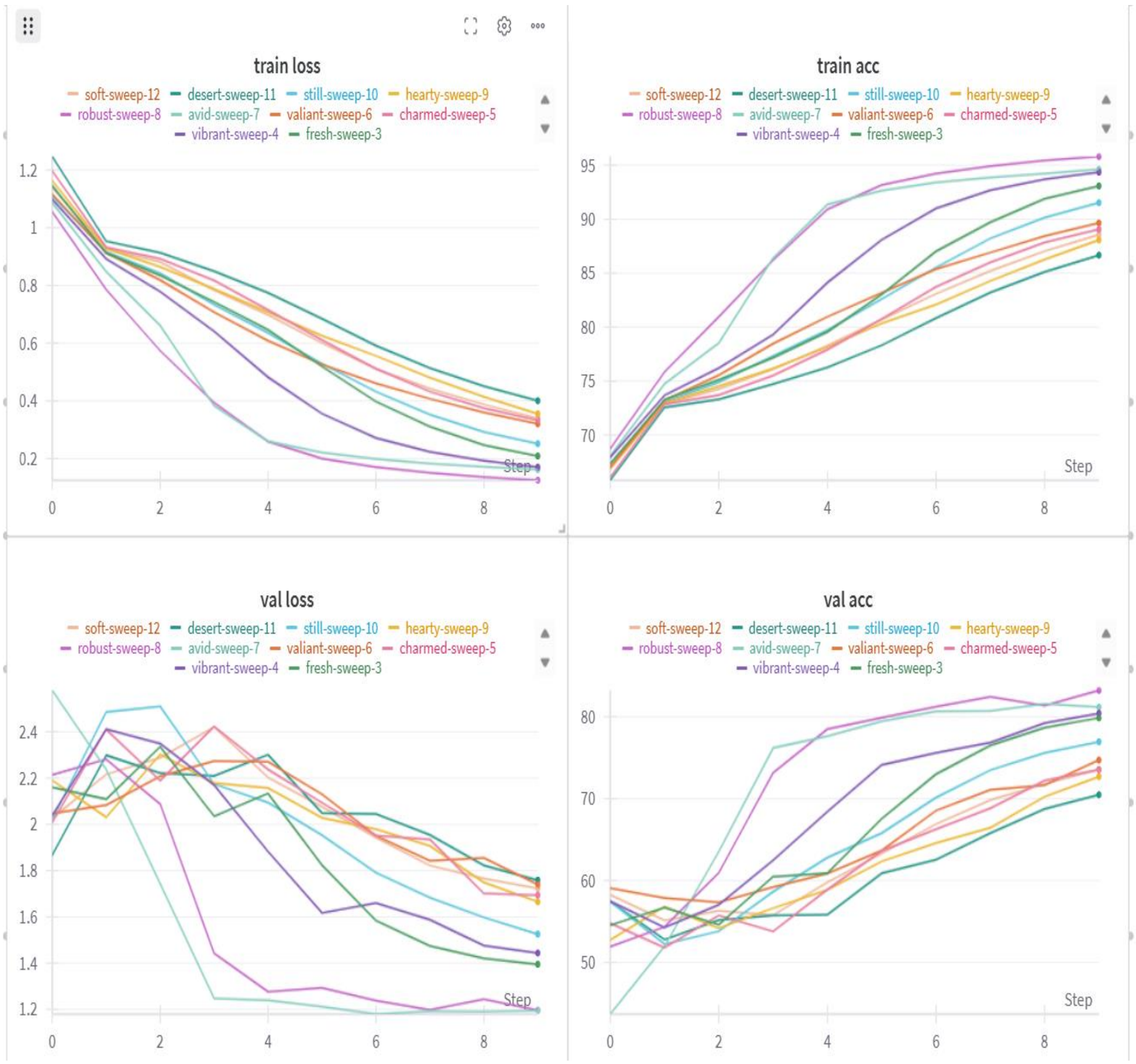
- Attention: ON
- Beam Search: OFF

Hyperparameters being tested in the sweep:

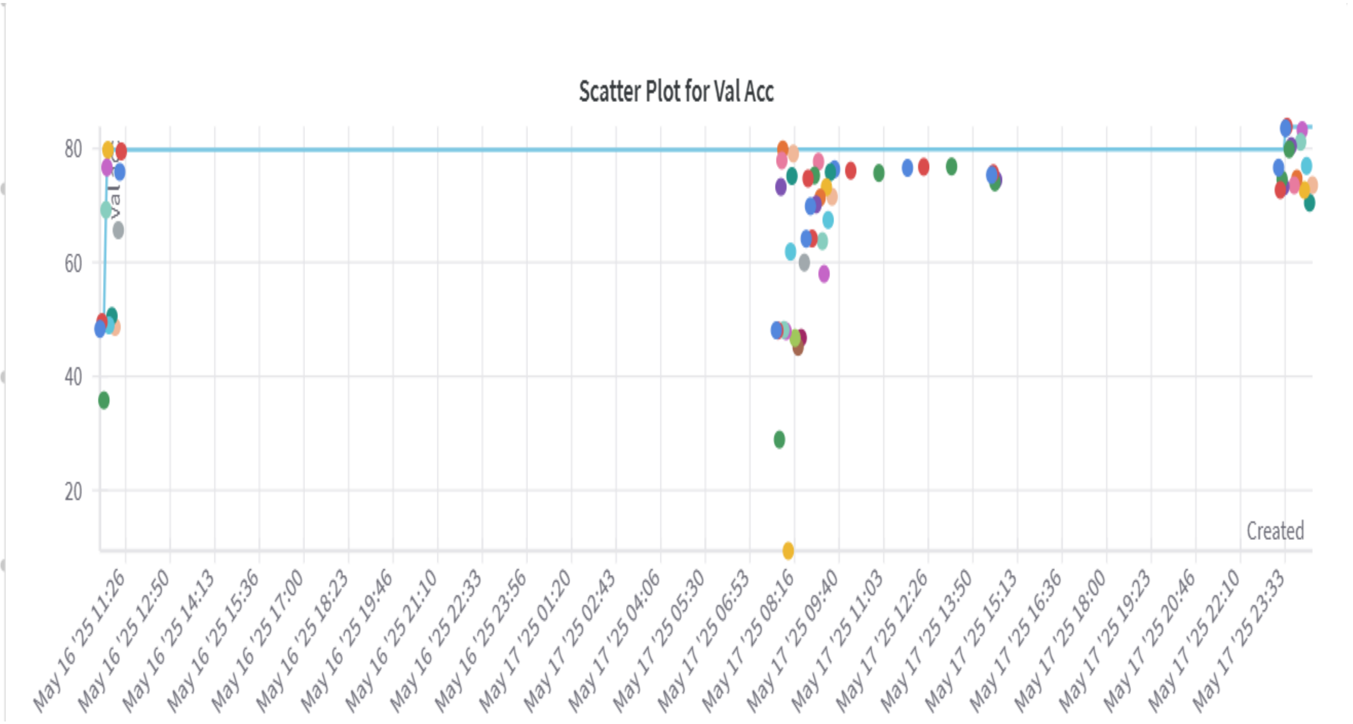
- Number of encoder and decoder layers: 1, 2, 3
- Hidden layer size: 128, 256
- Dropout: 0, 0.2

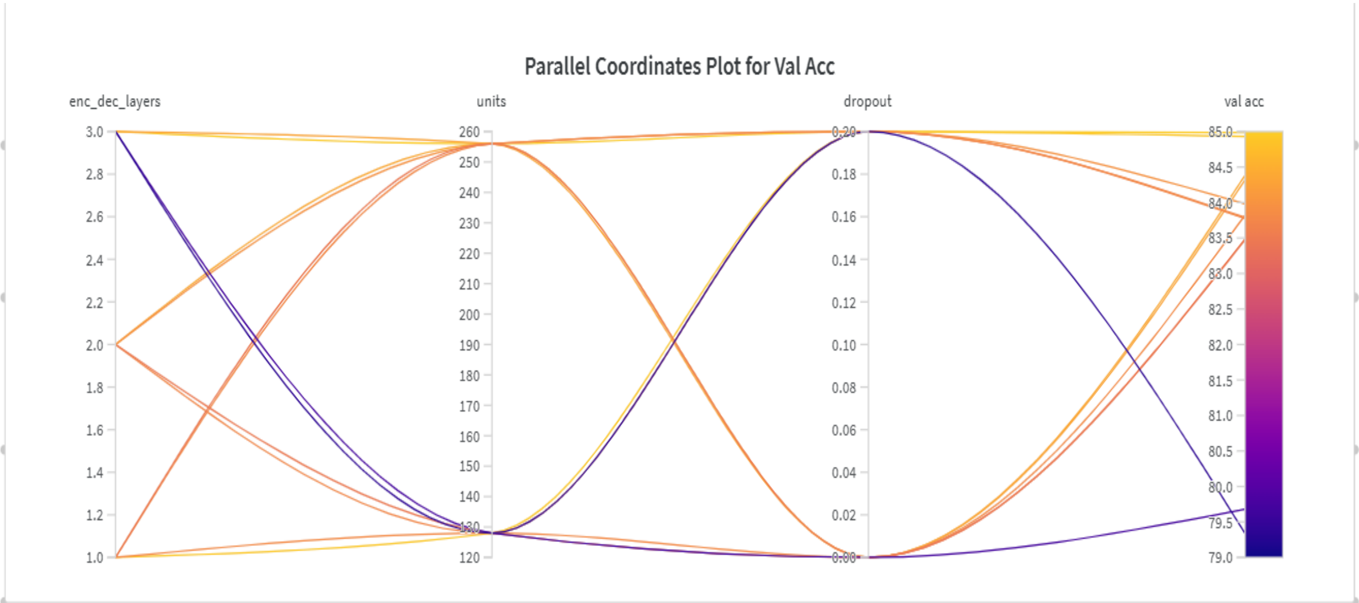
Sweep with Attention

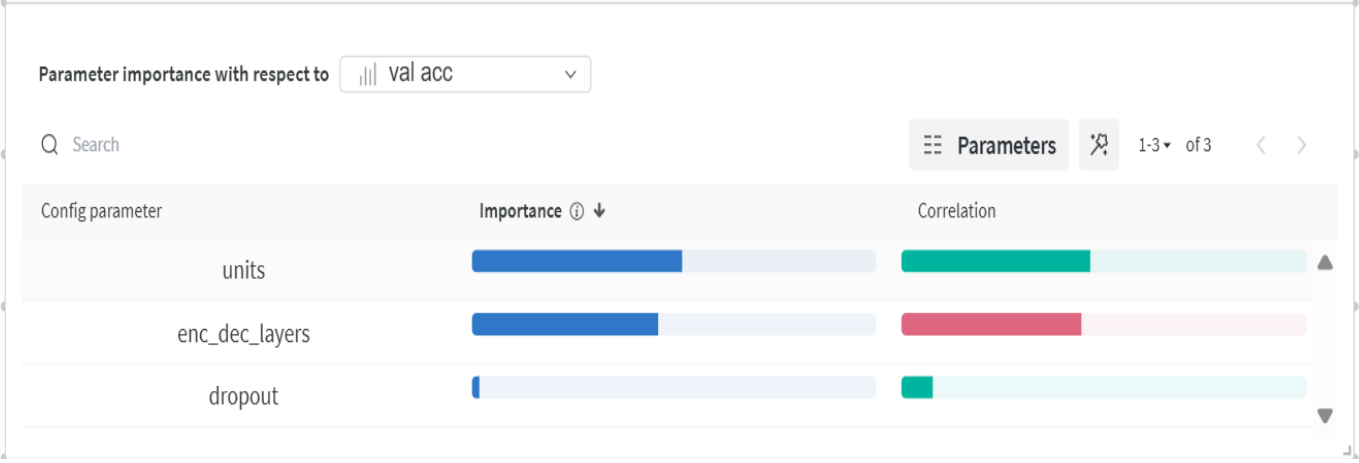
As stated above, this sweep tested the number of encoderdecoder layers, hidden layer size and dropout. The number of runs in this sweep are $3 \times 2 \times 2 = 12$. Each run is named as "layers{num_layers}_units{hiddensize}_drop{dropout}". The results of the sweep are as follows.



The scatter, parallel coordinates and parameter importance plots are given here.






Parameter importance with respect to val acc

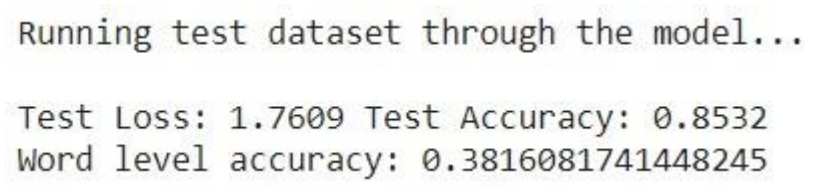
Config parameter	Importance ⓘ ↓	Correlation
units		
enc_dec_layers		
dropout		

b) Evaluating the Best Model

To evaluate the best model we run the best model on the test dataset. The best model was determined above using many sweeps and inferences from those sweeps. The configuration of the best model is:

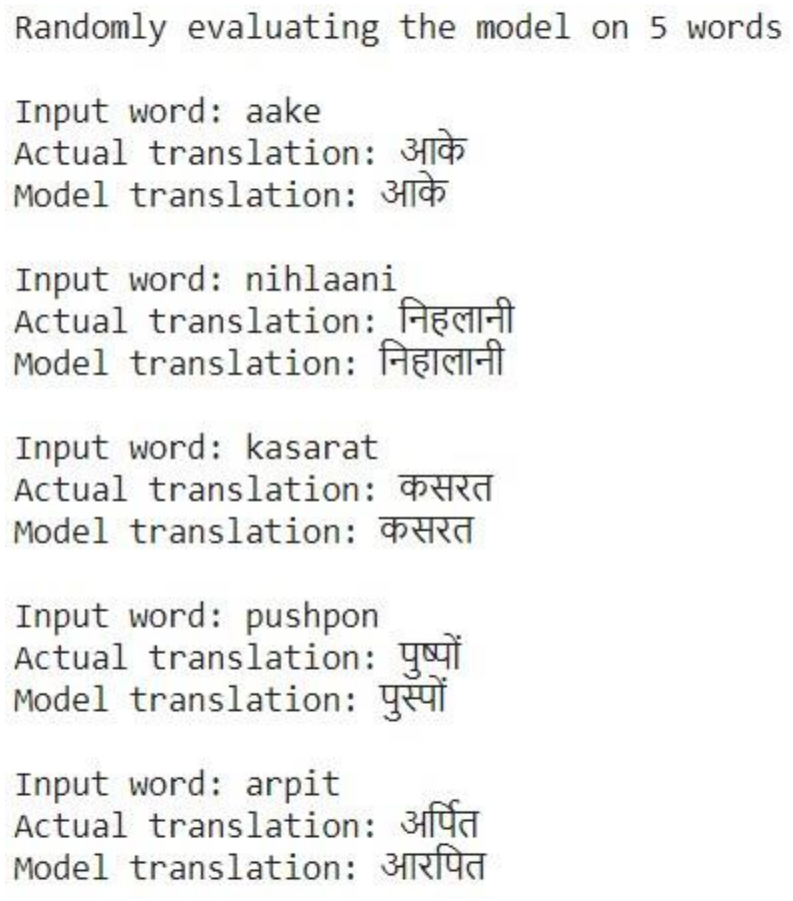
- Layer Type: LSTM
- Encoder and Decoder Layers: 3
- Embedding Dimension: 256
- Hidden Layer Size: 256
- Attention: ON
- Dropout: 0.2 Epochs:
- 30

The loss and accuracy on the test dataset were as follows.



```
Running test dataset through the model...

Test Loss: 1.7609 Test Accuracy: 0.8532
Word level accuracy: 0.3816081741448245
```

Here are some sample transliterations of the model.



```
Randomly evaluating the model on 5 words

Input word: aake
Actual translation: आके
Model translation: आके

Input word: nihlaani
Actual translation: निहलानी
Model translation: निहालानी

Input word: kasarat
Actual translation: कसरत
Model translation: कसरत

Input word: pushpon
Actual translation: पुष्पों
Model translation: पुस्पों

Input word: arpit
Actual translation: अर्पित
Model translation: आरपित
```

The predictions on the entire test set have been saved to "predictions_attention" and uploaded on the GitHub project.

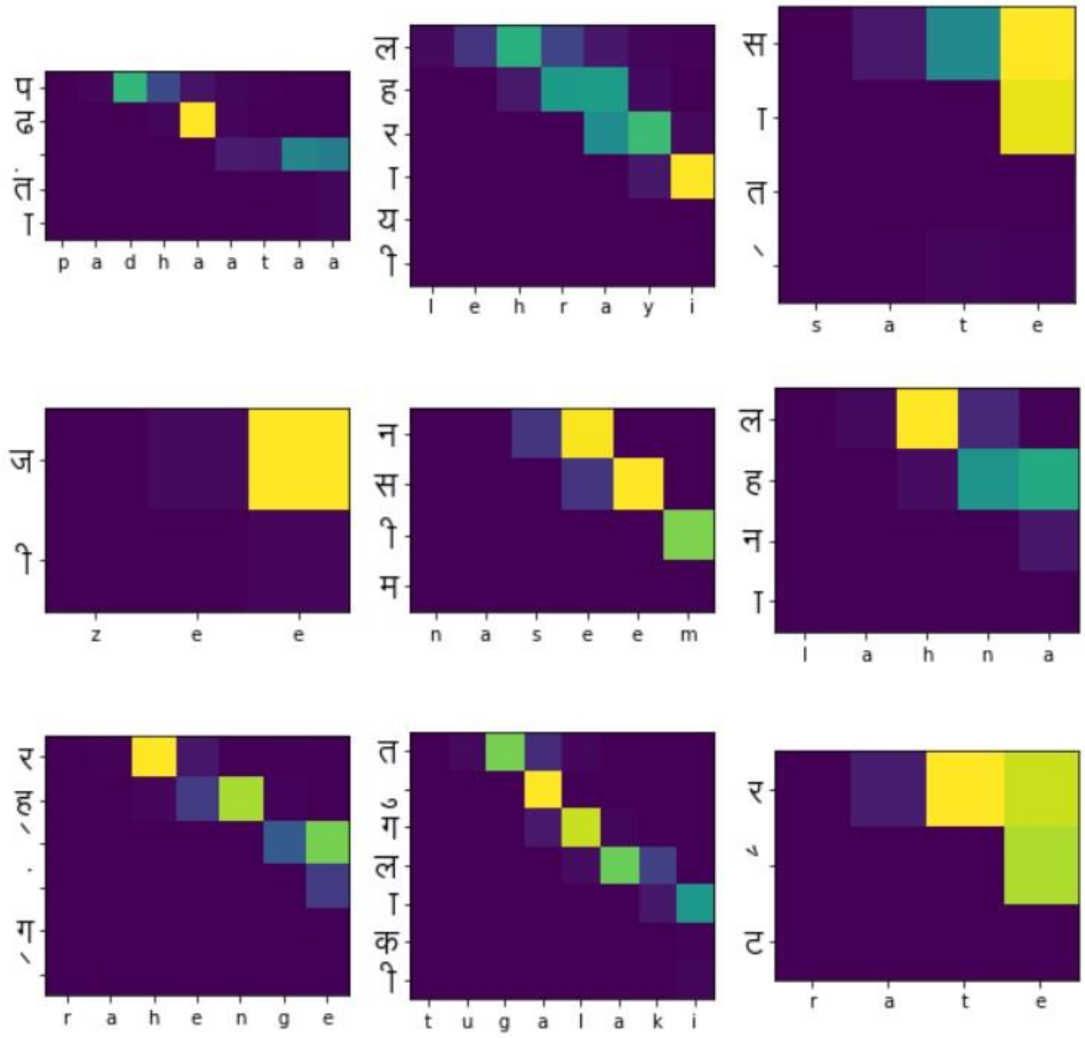
c) Inferences

- The main inference of note from this sweep is that attention, while useful for a sequence-to-sequence model, is not particularly effective here. The reason is that the task here is transliteration and not translation. In translation the difference in the structures of the input and target languages means that the model needs to look at different parts of the input while constructing each part of the output. However, in transliteration, since each input character (or consecutive input characters) largely map(s) to each output character(s) in the same order, attention isn't very effective for this task.
- As deduced earlier itself, LSTM is the best layer type. What we can confirm from this sweep is that a larger hidden layer size is better, as this can encode more complex information required for sequenceto-sequence tasks.
- 30 epochs may be a bit too much, as the training loss converges after about 15-20 epochs and the validation loss starts oscillating and gradually rising after that. However, it should be noted that the validation accuracy still gradually plateaus around 30 epochs so it may be fine.

- The trick to getting a good validation accuracy seems to be make the model as complex as possible (3 layers, 256 hidden units, 256 embedding size, etc.) and then apply regularization on top of it using dropout. Complex models with dropout had both good train and validation scores.
- The smoothness of training curves and the oscillations of validation curves indicates that the trend of doing better and better on the training data doesn't always map exactly to either doing better on validation data (underfitting) or to doing worse on validation data (overfitting), the correspondence is rather unpredictable.
- As can be seen from the word level accuracy, character-wise translation is very difficult as the entire word becomes wrong if even one output character is off. A word level accuracy of 38.1 percent is significantly better than the 34.8 percent achieved without attention, although the character level accuracy isn't much better (85.4 percent from 84.5 percent).

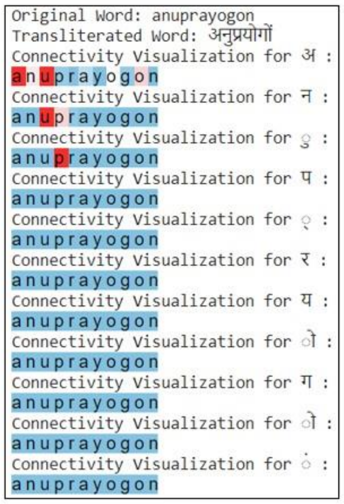
d) Attention Heatmaps

9 samples were taken from the test dataset and the attention weights corresponding to each character pair between each input and output character were plotted in a heatmap, to form a 3×3 grid of such heatmaps.

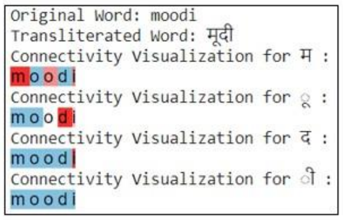


Question 6

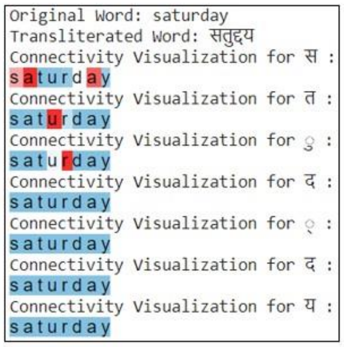
Using the starter code from the blog, we exactly mapped the connectivity between each output character and each input character. The gradients representing connectivity were in a list of length equal to that of the transliterated output, with each list element as a tensor of size (max_input_len, hidden_size). We took a norm along the hidden_size axis to get the magnitude of the gradient for each output character with respect to each input character. Then, the input word was printed once for each output character, with the gradient of that output character with respect to each input character determining the color behind that input character in the row for that output character. The color map used was a diverging colormap varying from blue (0) to white (0.5) to red (1).



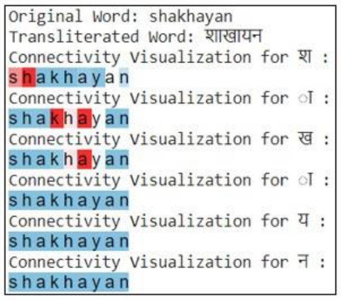
```
Original Word: anuprayogon
Transliterated Word: अनुप्रयोगों
Connectivity Visualization for अ :
anuprayogon
Connectivity Visualization for न :
anuprayogon
Connectivity Visualization for ु :
anuprayogon
Connectivity Visualization for प :
anuprayogon
Connectivity Visualization for ् :
anuprayogon
Connectivity Visualization for र :
anuprayogon
Connectivity Visualization for य :
anuprayogon
Connectivity Visualization for ो :
anuprayogon
Connectivity Visualization for ग :
anuprayogon
Connectivity Visualization for ो :
anuprayogon
Connectivity Visualization for ं :
anuprayogon
```



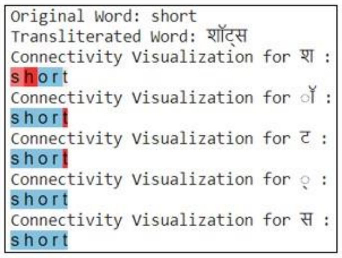
```
Original Word: moodi
Transliterated Word: मूदी
Connectivity Visualization for म :
moodi
Connectivity Visualization for ू :
moodi
Connectivity Visualization for द :
moodi
Connectivity Visualization for ी :
moodi
```



```
Original Word: saturday
Transliterated Word: सतुद्दय
Connectivity Visualization for स :
saturday
Connectivity Visualization for त :
saturday
Connectivity Visualization for ु :
saturday
Connectivity Visualization for द :
saturday
Connectivity Visualization for ् :
saturday
Connectivity Visualization for द :
saturday
Connectivity Visualization for य :
saturday
```



```
Original Word: shakhayan
Transliterated Word: शाखायन
Connectivity Visualization for श :
shakhayan
Connectivity Visualization for ा :
shakhayan
Connectivity Visualization for ख :
shakhayan
Connectivity Visualization for ा :
shakhayan
Connectivity Visualization for य :
shakhayan
Connectivity Visualization for न :
shakhayan
```



```
Original Word: short
Transliterated Word: शॉट्स
Connectivity Visualization for श :
short
Connectivity Visualization for ॉ :
short
Connectivity Visualization for ट :
short
Connectivity Visualization for ् :
short
Connectivity Visualization for स :
short
```

Note that the function mapping the gradient magnitude to the colour first went through a MinMaxScaler transform from the sci-kit learn library, in order to get the weights between 0 and 1. Hence if all the gradients are zero then all colors will be the same. Looking at the connectivity heatmaps, we can see that the gradients have a tendency to vanish for later timesteps of producing the output. Note that what one would expect is that for the first output character the model would look at the first 12 characters and the ones after that for the second output character and so on. This is how humans would approach a transliteration task. From the heatmaps we can see that the model looks at the first 1-2 characters for the first output character, but this approach largely vanishes later into the word as the gradients start vanishing. Later on the input characters focused on don't really match with where humans would look during transliteration, indicating that through Backpropagation Through Time (BPTT) the model has learnt some other way of carrying out this task.

Question 7

The link to the GitHub repository for this assignment is:

https://github.com/vamsikrishnamohan/DA6401--Assignment-3 The entire assignment has been done in one Kaggle notebook using GPU-accelerated runtimes.

Question 8(optional)

Not attempted.

Self Declaration

I, Susarla Vamsi krishna Mohan (Roll no: DA24M026), 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.

https://wandb.ai/da24m026-indian-institute-of-technology-madras/Assignment3/reports/DA24M026-Assignment-3---VmlldzoxMjc5OTg2Ng