Assignment 3

Use recurrent neural networks to build a transliteration system.

Argha Boksi cs21d407, Shubham Mallik Thakur

Created on April 14 | Last edited on May 8

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.
- We strongly recommend that you work on this assignment in a team of size 2. Both the members
  of the team are expected to work together (in a subsequent viva both members will be expected to answer questions, explain the code, etc).
- 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).

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 :

The code is flexible, in order to change the dimension of the input character embeddings, the cell (RNN, LSTM, GRU) and the number

of layers in the encoder and decoder one needs to update the "sweep_configuration" object in our implementation.

(a) Total Number of Computations :

$x_i = EI_i$, $s_i = \sigma(U x_i + W s_{i-1} + b)$, $y_i = \text{softmax}(V' s_i + c)$

Dimensions of the matrices :

$I_i : (V, 1)$, $E : (m, V)$, $U : (k, m)$, $W : (k, k)$, $s_{i-1} : (k, 1)$

$b : (k, 1)$, $V' : (k, k)$, $s_i : (k, 1)$, $c : (k, 1)$, $x_i : (m, 1)$

Encoder computations :

$s_i = \sigma(U x_i + W s_{i-1} + b)$

Multiplication : $O(km + k^2)$, Additions : $O(2k)$

Total : $O(km + k^2 + 2k)$

$EI_i : O(mV)$, Sigmoid : $O(k)$

Total number of computations performed per step for encoder :

$O(mk + k^2 + mV + 3k)$

Total number of computations performed for encoder for T steps =

$O(T(mk + k^2 + mV + 3k))$

Decoder computations :

$s_i = \sigma(U x_i + W s_{i-1} + b)$

Multiplication : $O(km + k^2)$, Additions : $O(2k)$

Total : $O(km + k^2 + 2k)$

$EI_i : O(mV)$, $y_i : O(Vk + V)$, softmax : $O(V)$

Total number of computations performed for decoder per step :

$O(mk + k^2 + mV + Vk + 2V + 2k)$

Total number of computations performed for decoder for T steps :

$O(T(mk + k^2 + mV + Vk + 2V + 2k))$

Total number of computations for encoder and decoder for T steps :

$O(T(2mk + 2mV + 2k^2 + Vk + 2V + 5k))$

(b)

Total Number of Parameters :

Number of Parameters for Encoder:

$E_1 : (m, V)$, $U : (k, m)$, $W : (k, k)$

$b : (k, 1)$, $V' : (k, k)$, $c : (k, 1)$

Total number of parameters for encoder :

$O(mV + km + 2k^2 + 2k)$

Number of Parameters for Decoder:

$E_2 : (m, V)$, $U : (k, m)$, $W : (k, k)$

$b : (k, 1)$, $V' : (V, k)$, $c : (V, 1)$

Total number of parameters for decoder :

$O(mV + km + k^2 + Vk + V + k)$

Total number of parameters :

$O(2mV + 2km + 3k^2 + 3k + Vk + V)$

Question 2 :

For the transliteration task we used **Hindi lexicons** from Google's Dakshina dataset.

We experimented with the following hyperparameter values :

cell type : LSTM, GRU, RNN

input embedding size : [16, 32, 64, 128, 256]

batch-size: [64, 128]

number of encoder layers: [1,2,3]

number of decoder layers: [1,2,3]

hidden layer size : [16, 32, 64, 128, 256]

number of epochs: 15

type of decoding: ['beam_search', 'greedy']

dropout probability : [0.2, 0.3] (We added recurrent dropout, via the "dropout" and "recurrent_dropout" arguments within the LSTM(), GRU() and SimpleRNN() functions)

learning rate: [1e-3, 1e-4]

beam width for beam search: [3, 5]

We obtained the highest validation accuracy**(37.60%)** for the following setting :

cell type: GRU

Input embedding size: 256

batch-size: 64

number of encoder layers: 2

number of decoder layers: 1

hidden layer size : 128

type of decoding: greedy

dropout probability : 0.3

learning rate: 0.001

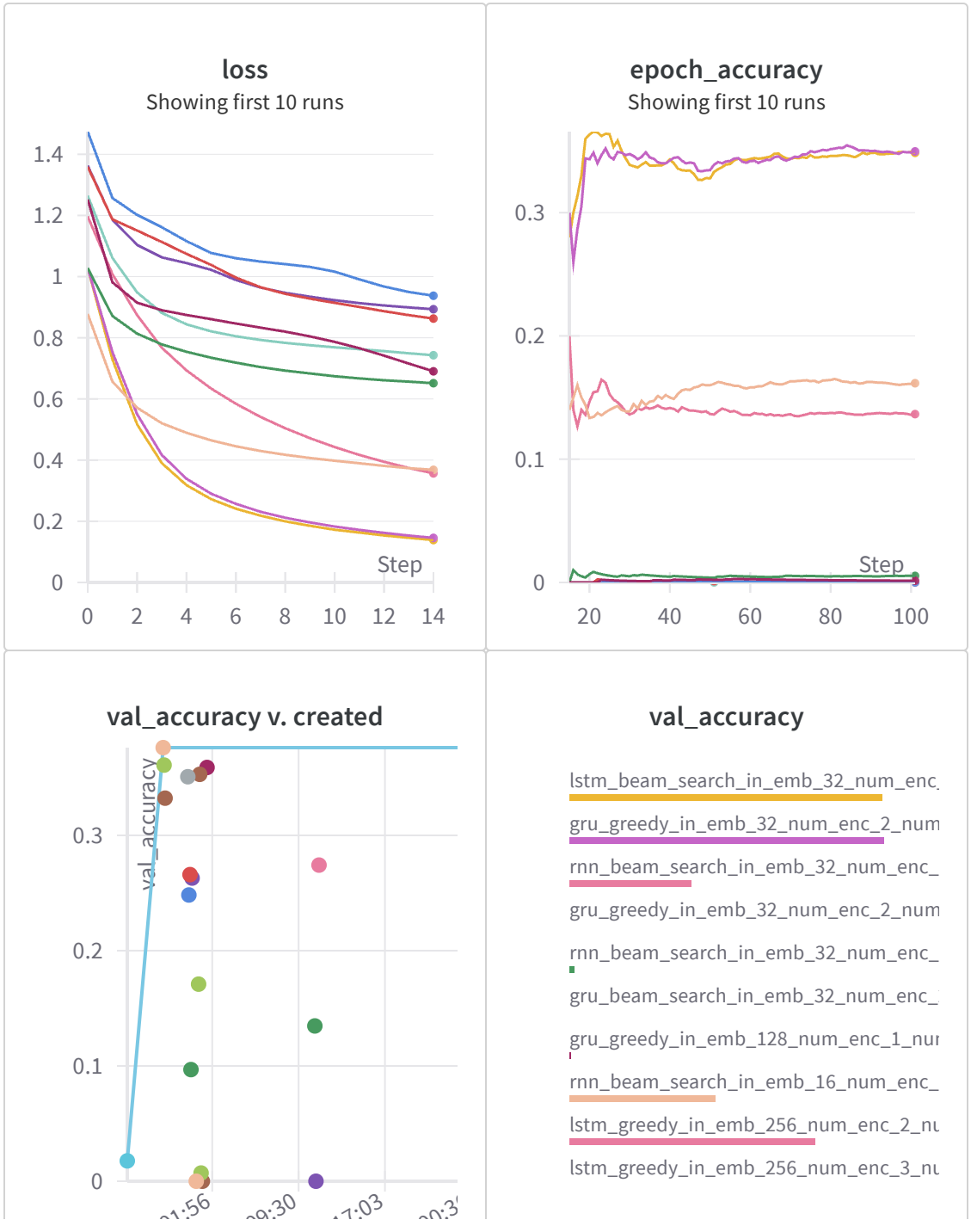
number of epochs : 15

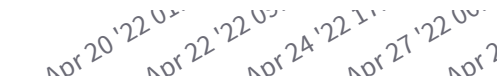
beam width for beam search : 3

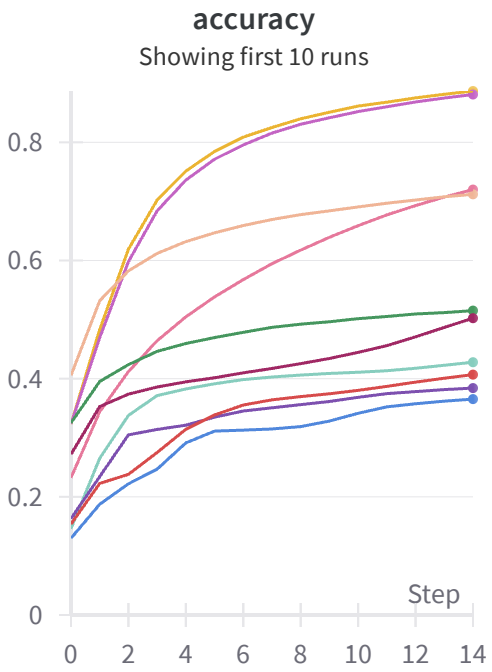
Every run takes more than 40 mins, so we kept the numbers of values for hyperparameters in the original sweep as small as possible. We used smaller sweeps for this purpose. For example the value of 16 for input embedding size was performing poorly

compared to other values and hence we removed that option in the original sweep. The smaller sweeps were used to adjust other hyperparameter values as well.

For efficient hyperparameter search we picked "Bayesian Search" method provided by wandb. "Bayesian Sweeps" create a probabilistic model of metric score as a function of the hyperparameters, and choose parameters with high probability of improving the metric(in our case the metric is "val_accuracy" and the goal is "maximize").







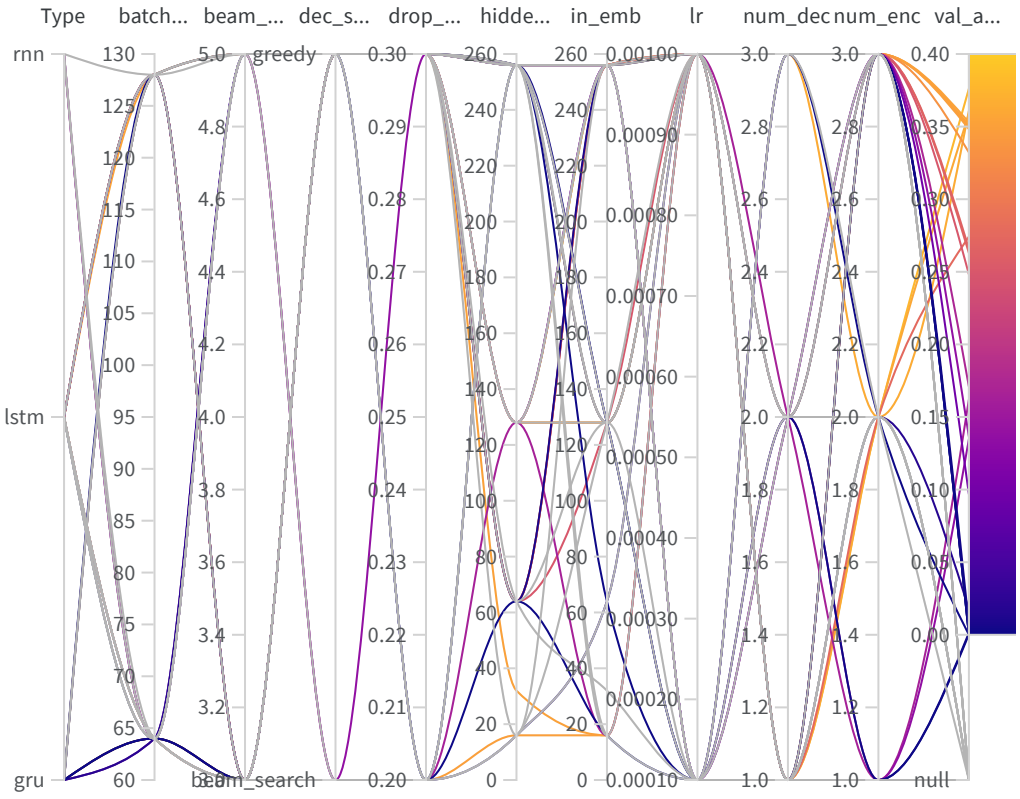
Parameter importance with respect to

val_accuracy

Search

Parameters

Config para…	Importance.	Correlati
lr		
drop_prob		
batch_size		
beam_wi…		



Question 3 :

1) All of our best models have a beam width of either 3 or 5. Hence it can be said that for this particular task having a beam width of greater than 1 is definitely an advantage. However we observed that increasing the beam width increased the time taken to train the model significantly.

2) Beam search with larger beam widths(3 or 5) perform better than those with smaller beam widths.

3) From the above plots it is clear that for the LSTM and GRU cell types, the validation accuracies are much higher than that of SimpleRNN. We observed that the performance of SimpleRNN did not improve even after adding more layers.

4) Among the three different cell types LSTM performed best on the validation data. LSTM's high positive correlation with validation accuracy reflects the same.

5) Almost all of our best models have "hidden layer size" = 128 or 256. Hence it can be said that adding more number of layers gave better performance for LSTM and GRU based models.

6) We observed that "Greedy Decoding" performed better during our experimentation. It is clear from the "val_accuracy v. created" plot that most of our best models use greedy decoding.

7) From the correlation summary table it is clear that "dropout probability" has a very high positive correlation with "validation accuracy". Almost all of our best models used "dropout probability = 0.3".

8) For all the best performing models "number of encoder layers" is equal to either 2 or 3. "number of encoder layers" has a high positive correlation with validation accuracy whereas "number of decoder layers" has a negative correlation.

9) Embedding size 16 and 32 gave very poor performance. It is clear from the correlation summary table that "input embedding size" has

a positive correlation with validation accuracy. "input embedding size" = 128/256 gave best validation accuracies.

10) As per the correlation summary table learning rate is the most important hyperparameter with respect to validation accuracy. It has a very high positive correlation as well. "learning rate = 0.001" with Adam optimizer gave best results.

11) "batch-size" has a positive correlation with validation accuracy. Batch sizes 64, 128 gave good results on validation data.

Question 4 :

(a) **Test Accuracy**: For our best model, we got an accuracy of **36.58%** on test dataset. This is word level accuracy, which means a prediction is considered correct only if the entire predicted word matches the target output.

(b) We have provided some interesting sample inputs from our generated predictions. Green symbolizes correct words, yellow for partially correct(possible answer due to vagueness in transliteration), orange for incorrect and red for issues in dataset.

Input	Target	Predicted
jing	जिंग	जिंग
vidhansabhaen	विधानसभाएं	विधानसभाएं
mumba	बंबा	मुंबा
vaishnodevi	वैष्णोदेवी	वैश्नोदेव
uthane	उठने	उठाने
bowling	बाउलिंग	बोलिंग
sitaram	सीताराम	सितारम
pardaa	पर्दा	परदा
tarkwaad	तर्कवाद	तर्कवाद
putaliyon	पुतलियों	पुतलियों

(c) The above examples are chosen from the generated file for predictions. From these examples we can make some interesting observations.

- Lets us look at the incorrectly predicted words marked in yellow. Despite being incorrect, their error is only in a few vowels, which changes only the pronunciation of the word a little. The words उठने and उठाने both seem reasonable transliterations of the word uthane. Similarly one can look at other examples such as pardaa, bowling etc. It is interesting to note that most incorrect words fall in this domain. There is hardly any error in consonants.
- We have observed that longer sequences are more likely to be incorrect. In the above example We can see that the yellow and orange words are mostly longer sequences. Having said so, we have also included specific examples from correctly predicted words which are longer in size. Though they do not seem to be in plenty.
- We have also found out what looks like some kind of anomalies with the dataset. Consider the word Mumba(marked in red) for example. Its target word is given as बंबा, but it does not seem correct. Infact we would consider the prediction of our model, मुंबा, to be a better candidate for the transliteration of that word. There seem to be several other examples of this sort in the dataset.

Question 5 :

We added an attention network to our basic seq2seq model and trained the model again. For the sake of simplicity, the attention network is added to a model with a single encoder and single decoder layer. We have used Bahdanau's attention.

(a) Hyperparameter tuning :

We obtained the highest validation accuracy of 47.06% for the following setting:

cell type: LSTM

input embedding size: 128

batch-size: 64

hidden layer size: 128

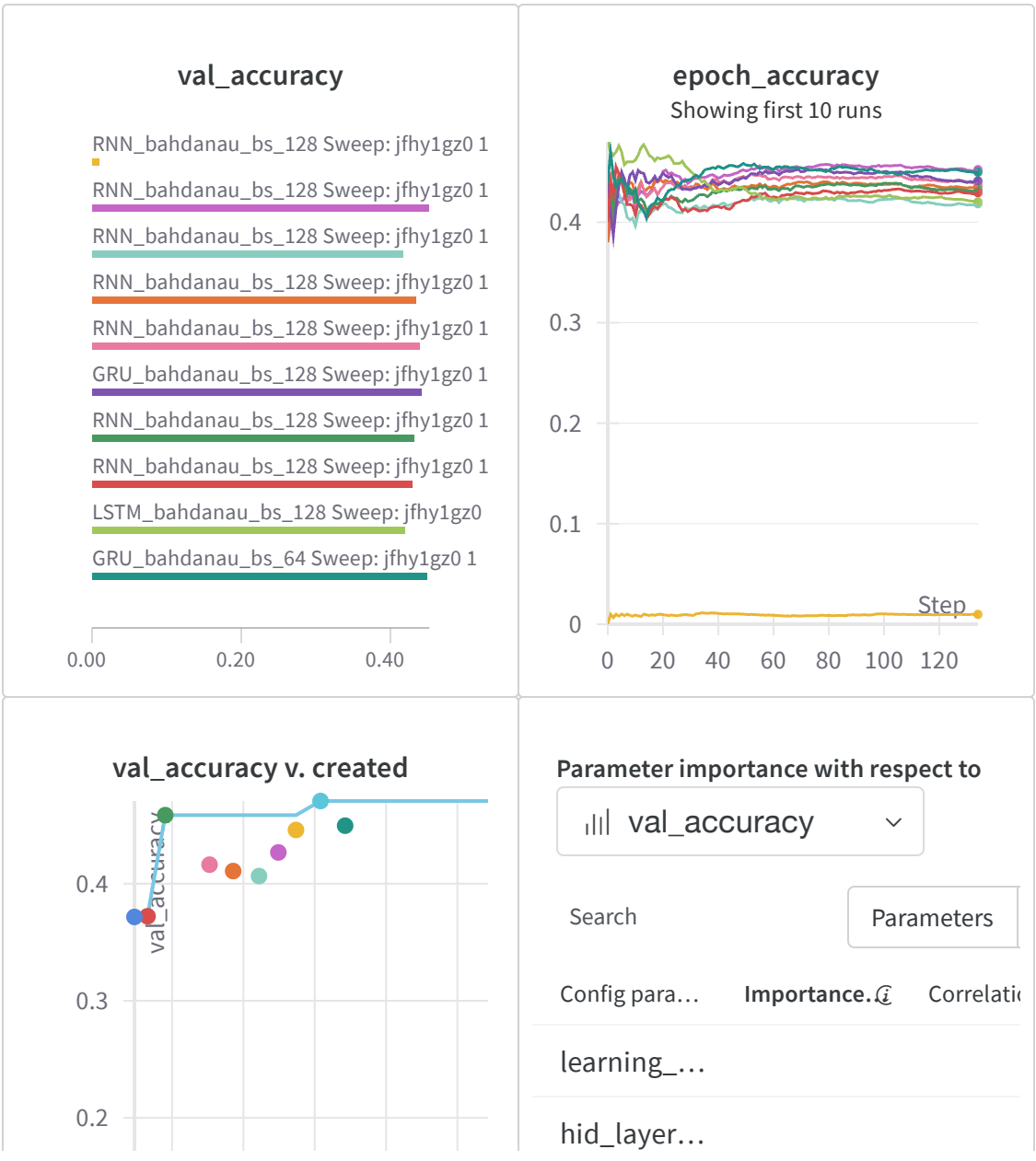
epochs: 15

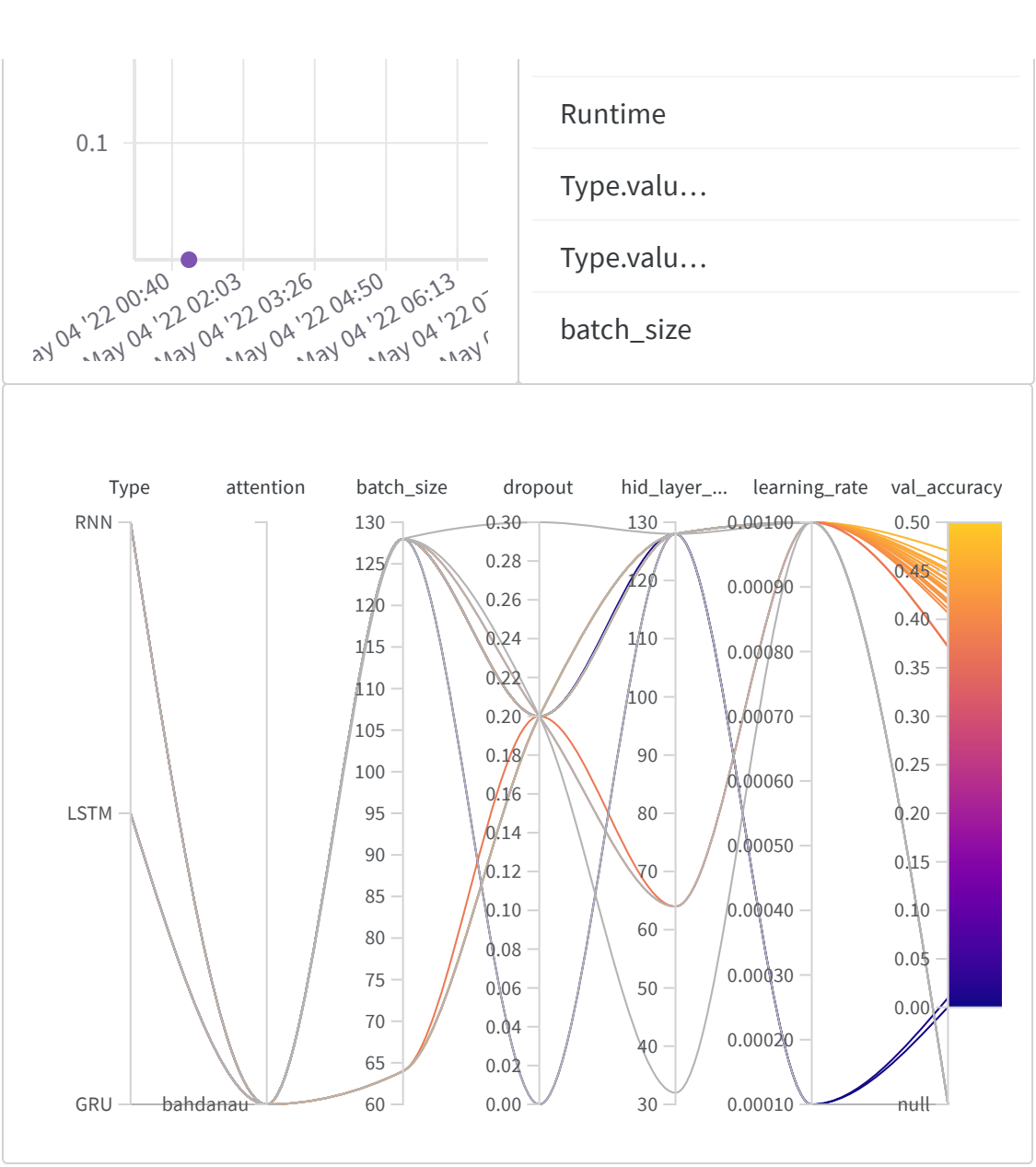
dropout: 0.2

learning rate: 0.001

attention type: Bahdanau

type of decoding : greedy





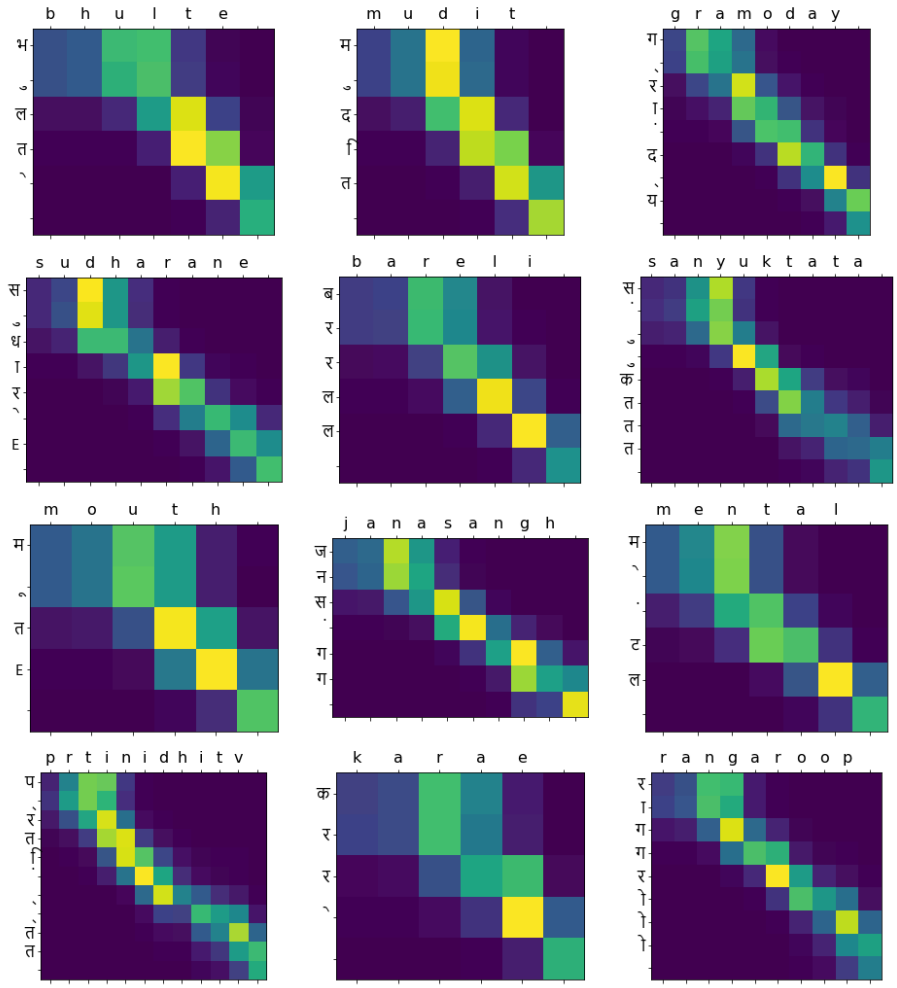
(b) The accuracy of our best model on test set is 40.79%. This is significantly better than our previous non-attention based model.

(c)

Input	Target	Vanilla	Attention
jasmeet	जसमीत	जस्मीत	जसमीत
namkeen	नमकीन	नमीकन	नमकीन
atmaen	आत्माएं	आतमाएं	आत्माएं
upakhyanon	उपाख्यानों	उपखानायों	उपाख्यानों
matsyavatar	मत्स्यावतार	मत्स्यत्वर	मत्स्यावतार

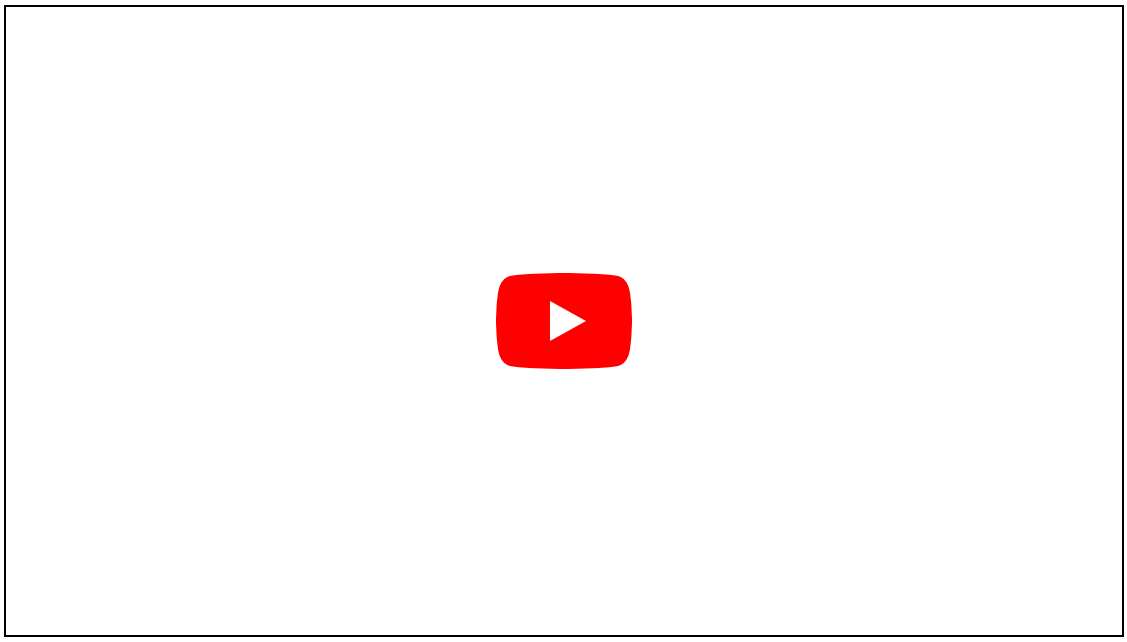
The attention model seems to perform better than the vanilla model, by about 5%. We looked at the prediction on the test set to determine where exactly this model performed better than the vanilla model. One can see from the test examples given above that certain words, with multiple vowels adjacent to each other(Higlighted in yellow), were incorrectly predicted by our vanilla model but inferred correctly by our attention model. Vanilla model also had some difficulty in working with longer sequences(Highlighted in orange), which were improved(but not entirely) to some extent by the attention model

(d) It is easily observed from the below heatmaps that shorter words tend to have attention distributed away from diagonal, while for longer words it is more centered around the diagonal.



Question 6 :

Given below is video describing the attention profile of model while decoding the word 'surakshit'. It was decoded as 'सुराषषत' and the video describes what part of input the model is looking at while decoding the ith character of the output. We collected the attention maps from the decoding process and then put them in the video, where ith frame describes the attention map while decoding ith character in output. Please refer to the code present in our github repo.



Question 7 :

Link to the GitHub code:

https://github.com/Shubhammt/cs6910_assignment3

Question 8 :

We finetuned the GPT2 model to generate lyrics for English songs. The code is present in the github repository.

Here are some of the songs generated by the GPT2 model -

1)

```
I love Deep Learning

[Chorus]

I love Deep Learning

[Chorus]

You're right, you're right, yeah, yeah, yeah, yeah, yeah, yeah, yeah, yeah
```

2)

```
I love Deep Learning, we just need some love in this room!

This time I'm going to try and be a great human being, this time I'm going to try and be a great human being

You've got to love this game, you've got to love this game

You've got to love this game, you've got to love this game

You've got to love this game, you've got to love this game
```

3)

```
I love Deep Learning, it's a little crazy for me, but I know how to make it work for me.
You can do it, you can do it, but it's crazy for me.
Deep Learning.
Deep learning, it's a little crazy for me, but I know how to make it work for me.
```

4)

```
I love Deep Learning
You can write and you can write

Deep Learning is how you learn to write

Deep Learning is how you learn to write
Deep Learning is how you learn to write
```

5)

```
I love Deep Learning and I'm so proud to be part of it
And I think it's funny, I think it's funny, I think it's funny
Oh you just made me laugh
I love Deep Learning and I'm so proud to be part of it
And I think it's funny, I think it's funny
Oh you just made me laugh
```

Self Declaration

CS21D407 : (50% contribution)

MA20M022: (50% contribution)

We, Argha Boksi and , Shubham Mallik Thakur swear on our honour that the above declaration is correct.

Created with ❤️ on Weights & Biases.

https://wandb.ai/shubham-argha/Assignment%203/reports/Assignment-3---VmlldzoxODQxMjE4