Assignment 3

Student Details

Name: Karan Agrawal

Roll No: CS24M021

WANDB Link:

https://wandb.ai/cs24m021-iit-madras/DA6401_A3/reports/Assignment-3--VmlldzoxMjg0MzgwNg?accessToken=lhh5nl2o7rdbjftu9gw8l4wl3pwvdv9hz3ce6s42wz4z6w5h8uuvlfa4a0hdgces

GitHub Link:

https://github.com/karan757527/DA6401_Assignment3/

Assignment 3

Use recurrent neural networks to build a transliteration system.

Karan Agrawal cs24m021

Created on May 19 | Last edited on May 20

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).

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)

Answer

Assume:

- Embedding size = m
- Hidden size = k
- Sequence length = T
- Vocabulary size = V
- Encoder and decoder are single-layered
- Using either RNN, LSTM, or GRU (see below for details)

We analyze computations per time step and multiply them by T to account for all steps:

- Encoder:

  Input embedding: mV multiplications

  Hidden state: $km + k^2$ multiplications, $k(m-1) + k(k-1) + 2k$ additions, k activations

- Decoder:

  Hidden state: $kV + k^2$ multiplications, $k(V-1) + k(k-1) + 2k$ additions, k activations

  Output: Vk multiplications, $V(k-1) + V$ additions, V softmax activations

Total computations:

Multiplications = $T(mV + km + k^2 + kV + k^2 + Vk)$

Additions = $T(m(V-1) + k(m-1) + k(k-1) + 2k + k(V-1) + k(k-1) + 2k + V(k-1) + V)$

(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)

Answer

Parameters are fixed and independent of the sequence length. We compute them per layer and sum:

- Encoder:
- Embedding matrix: mV
- RNN parameters: $km + k^2 + k$
- Decoder:
- Input projection: $kV + k^2 + k$
- Output projection: $Vk + V$

Total Parameters:

$= mV + (km + k^2 + k) + (kV + k^2 + k) + (Vk + V)$

$= mV + km + 2k^2 + 2k + kV + Vk + V$

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:

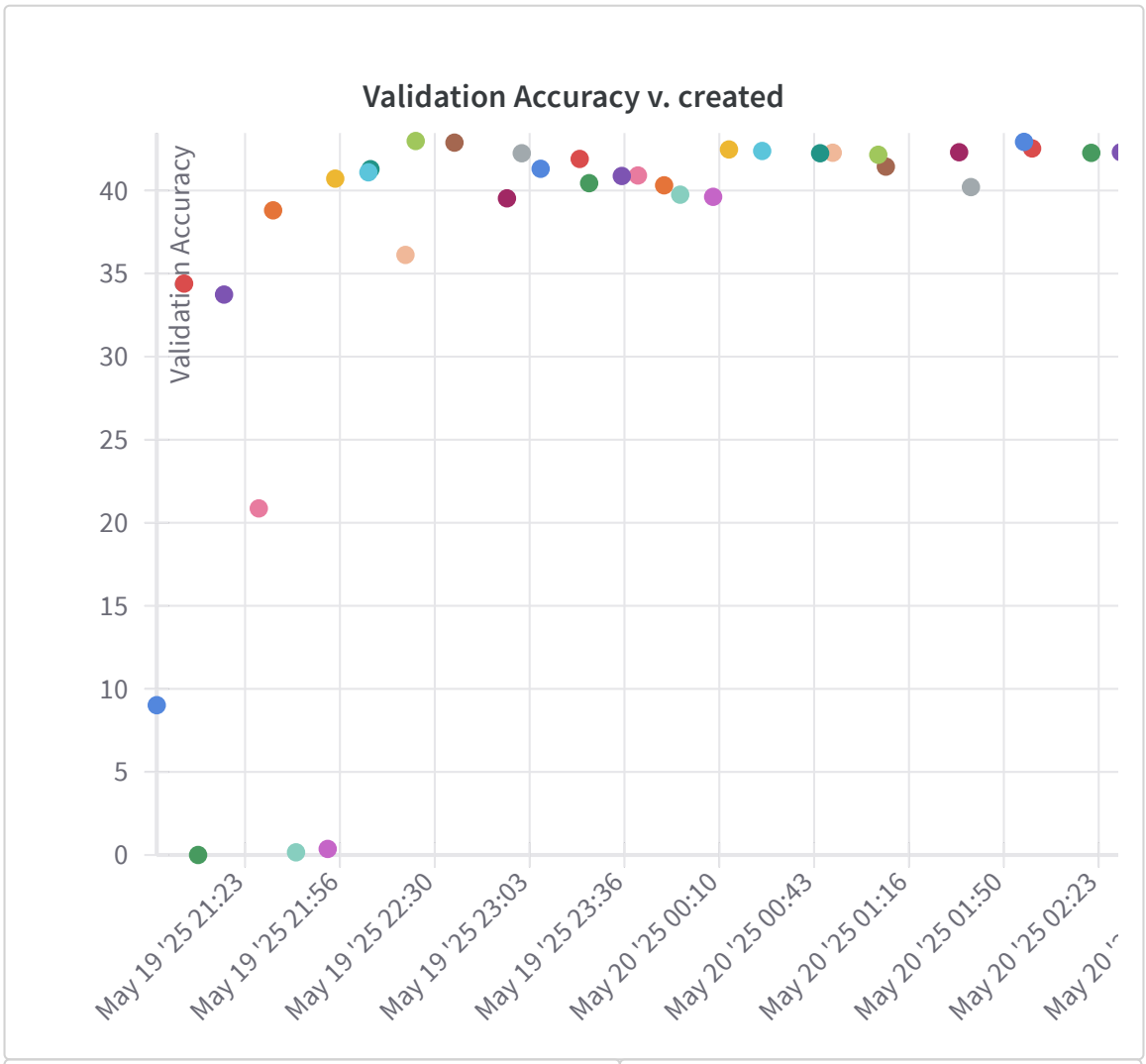
Based on your sweep please paste the following plots which are automatically generated by wandb:

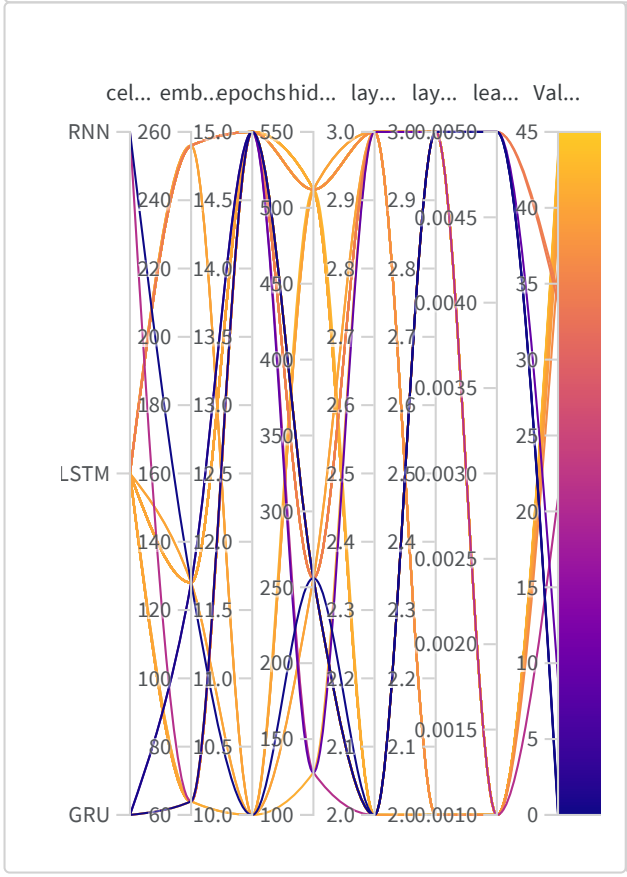
- accuracy v/s created plot (I would like to see the number of experiments you ran to get the best configuration).
- parallel co-ordinates plot
- correlation summary table (to see the correlation of each hyperparameter with the loss/accuracy)

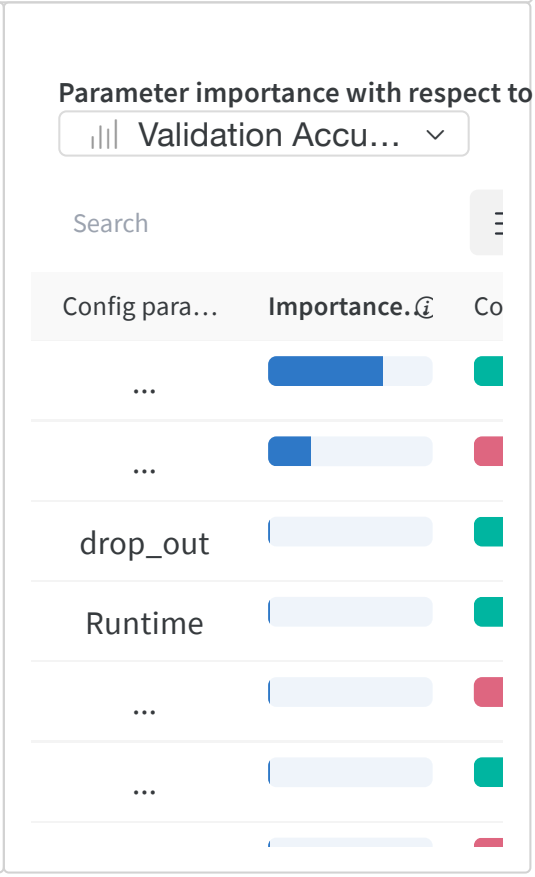
Also write down the hyperparameters and their values that you sweeped over. Smart strategies to reduce the number of runs while still achieving a high accuracy would be appreciated. Write down any unique strategy that you tried for efficiently searching the hyperparameters.

Answer

Plots:







To optimize the hyperparameters of our Seq2Seq transliteration

model trained on the Dakshina Hindi dataset, we used the Weights & Biases (wandb) sweep feature with a Bayesian optimization strategy.

- Sweep Strategy: Bayesian
- Objective: Maximize Character-Level Accuracy on the validation set
- Reasoning: Bayesian sweeps utilize a probabilistic model (typically Gaussian processes) to suggest promising hyperparameter combinations based on prior results, which helps reduce redundant or low-value experiments.

Efficiency Strategy: Early Stopping via Hyperband

To reduce training time and avoid resource wastage, we integrated early stopping using the Hyperband criterion:

- min_iter = 5: ensures each run is evaluated meaningfully.
- After ~8–10 epochs, the validation accuracy is compared to other trials.
- If the accuracy is significantly worse, the run is terminated early, preventing unnecessary full training.

The following hyperparameters were swept over:

- Embedding Size: [32, 64, 128, 256]
- Encoder Layers: [1, 2, 3]
- Decoder Layers: [1, 2, 3]
- Hidden Size: [64, 128, 256, 512]
- Recurrent Cell Type: ["rnn", "gru", "lstm"]
- Dropout Rate: [0.0, 0.2, 0.3]
- Batch Size: [32, 64, 128]
- Optimizer: ["adam", "nadam"]
- Epochs: [10, 15]

- Beam Search Size: [1, 3, 5]

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.

Answer

1. Cell Type

- The Vanilla RNN consistently underperformed.
- It had slower convergence, and even after 15 epochs, the validation and word-level accuracy remained significantly lower than GRU and LSTM.
- This is reflected in the negative correlation of the RNN cell type in the parameter importance chart.
- GRU emerged as the most efficient:
- It converged faster than both RNN and LSTM.
- Many of the top-performing models used GRU, indicating its robustness.

2. Hidden Layer Size

- This hyperparameter had the strongest positive correlation with model performance.
- Models with 512 hidden units consistently yielded higher validation and word-level accuracy.

- Smaller sizes like 32 or 64 underperformed and were quickly deprioritized during the sweep.

3. Embedding Size

- Higher embedding dimensions (256, 512) led to better results.
- Models with 32-dimensional embeddings were among the poorest performers and were excluded early by Bayesian pruning.
- There was a positive correlation between embedding size and validation accuracy.

4. Batch Size

- Models trained with batch sizes of 64 and 128 performed significantly better than those with 32.
- Batch size = 128 was dominant among high-accuracy runs in the parallel coordinates plot.

5. Optimizer

- NADAM clearly outperformed ADAM.
- It showed a strong positive correlation with validation accuracy in the correlation summary.
- Most high-performing configurations used NADAM, and they converged faster.

6. Encoder and Decoder Layers

- Encoder Layers:
- Deeper encoder stacks (e.g., 5 layers) generally degraded performance.
- 1 or 3 layers worked best; hence, this parameter had a mild negative correlation.
- Decoder Layers:
- No strong trend, but 2 or 3 layers sometimes showed marginal gains.

- The correlation was weak, suggesting a lesser impact compared to other parameters.

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).

Answer

The best-performing model selected based on validation set experiments had the following configuration:

- Batch Size: 32
- Cell Type: “LSTM”
- Dropout: 0.5
- Embedding Dimension: 64
- Encoder Layers: 3
- Decoder Layers: 3
- Hidden Size: 256
- Optimizer: Nadam
- Epochs Trained: 15
- Validation Accuracy (Word-Level): 43.46 %

This configuration yielded a word-level validation accuracy of 43.46%. The same model, when evaluated on the test set, achieved a word-level accuracy of 38.54%.

(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.

Answer

Here are sample predictions:

```
runs.summary["Prediction Table"]
```

Filter

	Input Word	Target Word	Predicted Word
2	anka	अंक	अंका
3	ankit	अंकित	अंकित
4	anakon	अंकों	अनाकों
5	ankhon	अंकों	अंखों
6	ankon	अंकों	एंकों
7	angkor	अंकोर	एंग्कोर
8	ankor	अंकोर	एंकोर
9	angaarak	अंगारक	अंगारक
10	angarak	अंगारक	अंगरक

1 - 10 of 32 Export as CSV Columns... Reset ta

Uploaded prediction_vanilla.csv on GitHub.

(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 may be it's just me!
- ...

Answer

- Transliteration accuracy significantly decreases for longer input

sequences. For example, the model transliterates badhachadhakar incorrectly as बढ़ाधकार (correct: बढ़ाचढ़ाकर) and vishwasniiyataa as विश्वनियत्ता (correct: विश्वसनीयता).

- The model struggles particularly with transliterating vowels 'a' and 'aa' to अ and आ, respectively, due to context dependence. For instance, it incorrectly transliterates anka to अंका rather than अंक.

- It frequently confuses similar-sounding character pairs such as (क्ष, श), (च, छ), and (न, ण).

- Errors related to short vowels (Rhaswa) and long vowels (Dirgha) occur regularly. Additionally, conjunct consonants are not predicted well; for example, chennai is transliterated as चेननाई instead of चेन्नई.

- The predictions are highly sensitive to English vowel changes. Minor typos significantly alter outputs, as seen when gayatri (correctly transliterated as गायत्री) changes to gayitri, resulting in an incorrect transliteration गायतिरी. Similarly, gaytri, a probable typo, incorrectly becomes गयत्री.

- Real-world inputs frequently contain typos; hence, models should be robust enough to handle minor input errors gracefully.

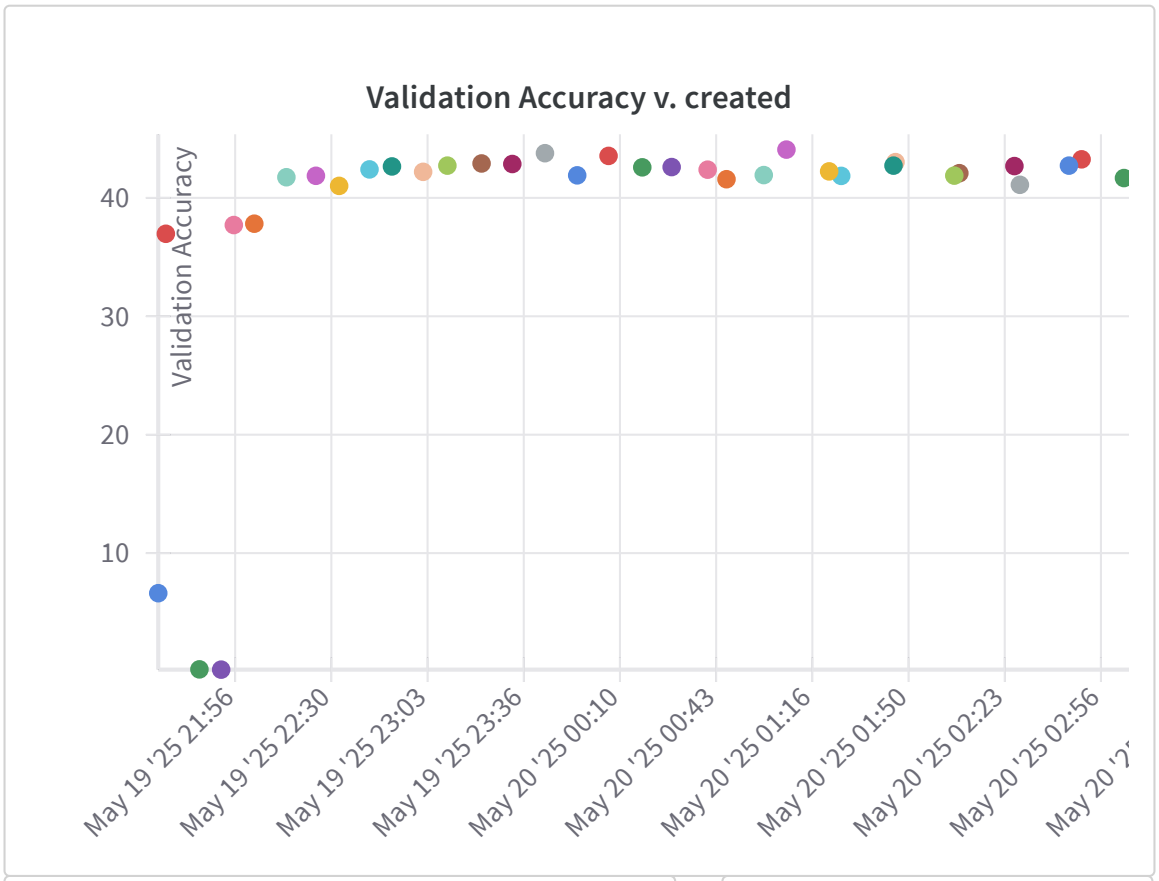
Question 5 (20 Marks)

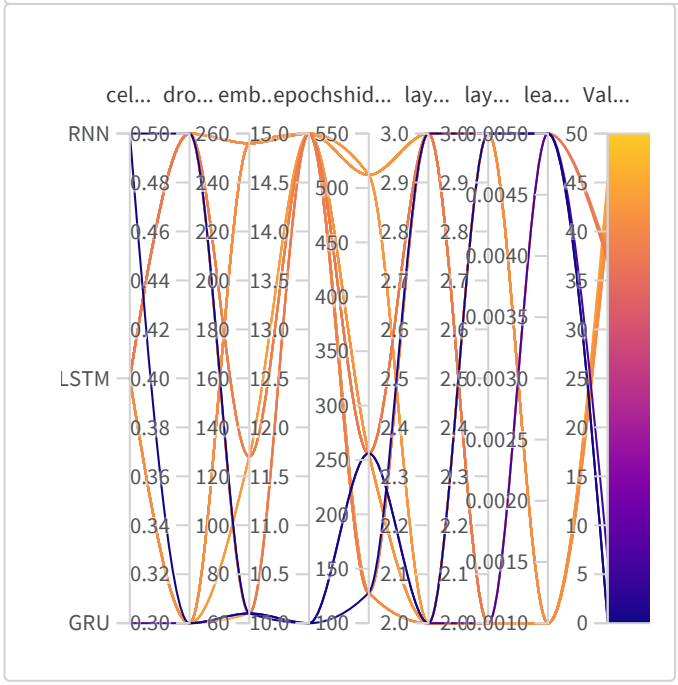
Now add an attention network to your basis sequence to sequence model and train the model again. For the sake of simplicity you can use a single layered encoder and a single layered decoder (if you want you can use multiple layers also). Please answer the following questions:

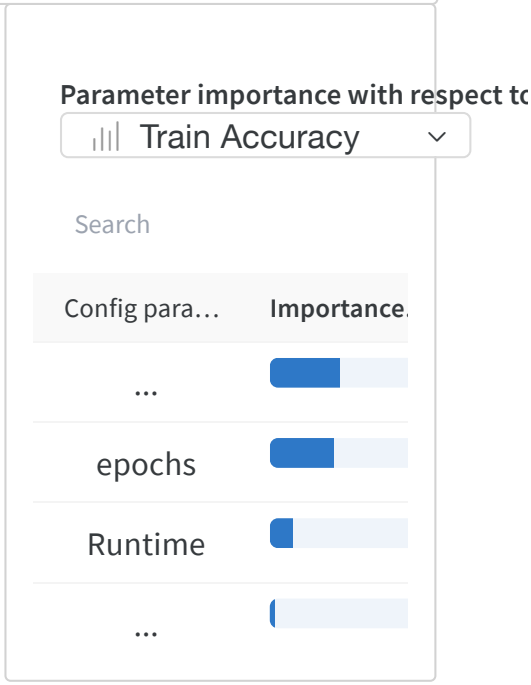
(a) Did you tune the hyperparameters again? If yes please paste appropriate plots below.

Answer

Yes, I changed the hyperparameters again and here are the plots:







(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.

Answer

The best-performing model selected based on validation set experiments had the following configuration:

- Batch Size: 16
- Cell Type: “LSTM”
- Dropout: 0.5
- Embedding Dimension: 64
- Encoder Layers: 3
- Decoder Layers: 3
- Hidden Size: 512
- Optimizer: Nadam
- Epochs Trained: 15
- Validation Accuracy (Word-Level): 45.36%
- Test Accuracy (Word-Level): 41.04%

This configuration yielded a word-level validation accuracy of 45.36%. The same model, when evaluated on the test set, achieved a word-level accuracy of 41.04%.

Also, Predictions on the test set are uploaded as prediction_attention.csv on Github.

(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)

Answer

The Attention model successfully resolves several shortcomings of the Vanilla model, as evidenced by an improved accuracy on the test set despite having fewer layers

The Attention mechanism notably improves transliterations by better recognizing subtle distinctions between characters, handling missing vowels, and avoiding incorrect character insertions or omissions.

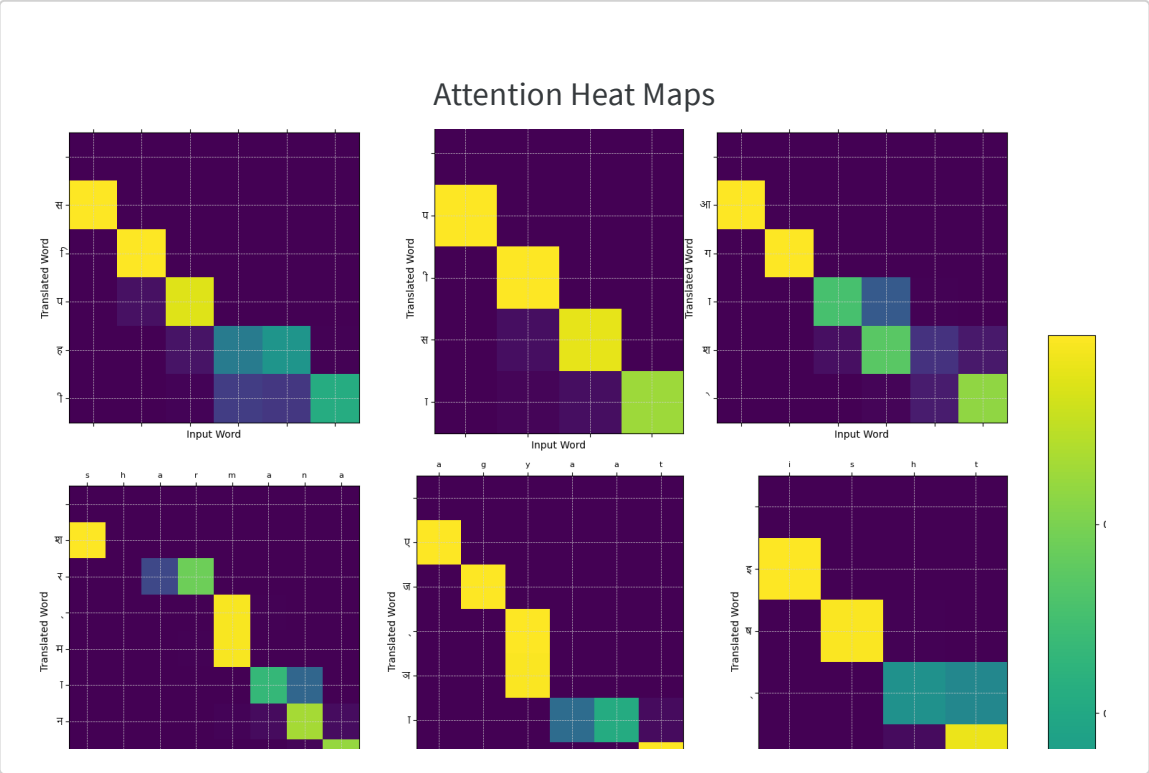
For Example:

English Word Transliteration	Vanilla Prediction	Attention Prediction	True
achyut	अछुत	अच्युत	अच्युत
lbdh	लुब्धी	लब्ध	लब्ध

In the example of achyut, the Vanilla model incorrectly transliterated 'च्य' as 'छु', which the Attention model accurately corrected. Similarly, for lbdh, the Vanilla model incorrectly introduced an additional vowel 'ु' and an extra vowel-ending 'ी', while the Attention model provided the correct transliteration.

Despite these improvements, the Attention model isn't flawless. For instance, a longer and complex word such as badhachadhakar was incorrectly transliterated by the Attention model as बधाचाधकार instead of the accurate बढ़ाचढ़ाकर. However, compared to the Vanilla model's prediction (बढ़ाधकार), the Attention model shows fewer character omissions, making it more reliable as it preserves more of the original meaning. Such improvements highlight the Attention model's ability to mitigate significant transliteration errors like character dropping, although spelling errors may persist.

(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).



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!

Answer

Sample ID	Transliteration	Char-level Focus Map
247	kartik → **कार्टिक**	का k a र् k a r टि a r t i क t i k

Color Interpretation:

- Dark Red (rgb(139,0,0)): High attention — the Latin character

contributed significantly to predicting the corresponding native character.

- Dark Teal (rgb(0,128,128)): Moderate attention — the Latin character contributed somewhat, though not dominantly.
- Sky Blue (rgb(173,216,230)): Minimal or no attention — the character had little to no influence on predicting the native character.

Character-wise Focus Observations:

- का
- 'a' receives high attention (dark red), which is expected as it directly maps to आ.
- 'k' is assigned moderate focus (teal), showing partial contribution — though ideally it should be stronger.
- र्
- 'a' again gets high focus, but 'r' is surprisingly only moderate — a subtle modeling error, suggesting some misalignment or distributed attention.
- 'k' and 't' are barely involved (sky blue), as expected.
- टि
- 'r' is the main contributor here (dark red), but 't' and 'i' receive only medium/low weight, which is suboptimal — 't' should dominate this segment.
- 'a' gets unnecessary moderate focus, reflecting some confusion in the attention map.
- क
- 'k' finally gets its deserved high weight (dark red) as the final consonant.
- 't' and 'i' show moderate influence, hinting at some blending in the attention assignment.

Question 7 (10 Marks)

Paste a link to your github code for Part A

https://github.com/karan757527/DA6401_Assignment3

Self Declaration

I, Karan Agrawal (Roll no: CS24M021), 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.

Created with ❤️ on Weights & Biases.

https://wandb.ai/cs24m021-iit-madras/DA6401_A3/reports/Assignment-3--VmlldzoxMjg0MzgwNg