Assignment - 3

Sequence to sequence model with & without Attention mechanism Analysis

Student Details:

Name : Indra Mandal

Roll No : ED24S014

Wandb Link: Wandb Report

Github Repository Link: Github Repo

Assignment 3 Seq2seq Model:

Use recurrent neural networks to build a transliteration system.

Indra Mandal ed24s014

Created on May 13 | 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).

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)

Answer:

Q.1/Part-1: Flexible code 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.

>>

I have implemented a flexible RNN-based Seq2Seq model for character-level translation . The model consists of:

- *Input Layer: Character embedding layer to convert input characters into dense vectors.*
- *Encoder: A configurable RNN (supports RNN, LSTM, GRU) that encodes the input sequence into a context vector. The number of layers, hidden size, and bidirectionality are adjustable.*
- *Decoder: A similar RNN-based module that takes the encoder's final state and generates one output character at a time. It supports teacher forcing and beam search decoding.*
- *Flexibility: The model design allows customization of embedding size, hidden size, number of encoder/decoder layers, dropout, RNN cell type, and decoding strategy (greedy or beam search).*

This setup is fully modular and supports easy experimentation with different hyperparameters via a single calling function.

```
# Set beam width (1 = greedy decoding, >1 = beam search)
beam_width = 3
beam_search = beam_width > 1

# Initialize the model
model = Build_Model(
    sequence_data_preprocessor=SequenceDataPreprocessor,
    encoder=Encoder,
    decoder=Decoder,
    seq2seq=Sequence2Sequence,
    batch_size=32,
    train_path=train_df,
    val_path=dev_df,
    device=device
)

# Build and train the model
seq2seq, train_model, evaluate_model, loss_acc_logs, _ = model.build(
    emb_size=64,
    layer_type="rnn",           # Type of Layers LSTM, GRU, RNN
    hidden_layers_size=512,
    num_encod_layers=3,         # Number of Decoder layers
    num_decod_layers=3,         # Number of Enoder layers
    dropout_rate=0.3,           # Percent of epochs
    epochs=30,                  # No of epochs
    learning_rate=0.0001,       # Add learning rate
    teacher_force_ratio= 1,     # Add percent of Teacher forcing which will decay exponationally a
    bidirectional=True,         # Set to True for Bidirectional
    patience=3,                 # Patience for Early Stopping
    val_beam_search= True,      # Use beam search for validation
    beam_width= 3,              # Add the beam width
    testing_phase= True,        # Set to True for testing
    test_path=test_df,
    test_beam_search=True,      # Use beam search for testing
    wandb_log = False
)
```

Q.1/Part-2: Total number of Computations and Total number of Parameters in the RNN based seq2seq model

➤ *Total Number of Computations in the Network*

- *Input embedding size:* m
- *Hidden state size:* k
- *Sequence length:* T *(same for input and output)*
- *Vocabulary size:* V
- *Encoder and decoder: 1 layer each*

1. Embedding Layer

For both encoder (input) and decoder (output), each character is mapped to an m-dimensional embedding. For a sequence of length T:

- *Encoder:* T *embeddings, each of size* m
- *Decoder:* T *embeddings, each of size* m

But embedding lookup is not a matrix multiplication, so it doesn't count as a major computation (just a table lookup).

2. Encoder RNN Computations

For a simple RNN cell at each time step, the main computation is:

$$\boxed{h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + b_h)}$$

Where:

- x_t *is the embedding at time t (m dimensional)*
- h_{t-1} *is the previous hidden state (k dimensional)*
- W_{xh} *is* $k \times m$
- W_{hh} *is* $k \times k$
- b_h *is* k *dimensional*

Each time step involves:

- $W_{xh}x_t$ *:* $k \times m$ *matrix times* m*-vector =* $k \times m$ *multiplications*

- $W_{hh}h_{t-1}$: $k \times k$ *matrix times* k*-vector =* $k \times k$ *multiplications*

Total per time step: $k \times m + k \times k$ *multiplications*

For T *time steps:* $T \times (k \times m + k \times k$

3. Decoder RNN Computations

Same as encoder, as the structure is the same.

For T *time steps:* $T \times (k \times m + k \times k)$

4. Output Layer (per time step in decoder)

The decoder output at each time step is projected to the vocabulary size:

$$\boxed{y_t = softmax(W_{hy}h_t + b_y)}$$

Where W_{hy} *is :* $V \times k$.

- *Each time step:* $V \times k$ *multiplications*
- *For* T *times steps:* $T \times V \times k$

5. Total Computations

Sum up all major matrix multiplications:

Total Computations = Encoder RNN + Decoder RNN + Output Layer

$$= T(km + k^2) + T(km + k^2) + T(Vk)$$
$$= 2T(km + k^2) + T(Vk)$$

Total number of computations per input-output sequence:

$$\boxed{2T(km + k^2) + T(Vk)}$$

➤ *Total Number of Parameters in the Network*

1. Embedding Layer

- *Encoder embedding:* $V \times m$
- *Decoder embedding:* $V \times m$

2. Encoder RNN Computations

For a simple RNN cell:

- W_{xh} *:* $k \times m$
- W_{hh} *:* $k \times k$
- b_h *:* k

Total: $k \times m + k \times k + k$

3. Decoder RNN Computations

Same as encoder RNN.

Total: $k \times m + k \times k + k$

4. Output Layer

- W_{hy} *:* $V \times k.$
- $b_y : V$

5. Total Parameters

Sum all parameters:

Total Parameters = Encoder Embedding + Decoder Embedding + Encoder RNN + Decoder RNN + Output Layer

$$= V \times m + V \times m + (k \times m + k \times k \times k) + (k \times m + k \times k + k) + (V \times k + V)$$

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

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

Total number of parameters in the network:

$$\boxed{2Vm + 2km + 2k^2 + 2k + Vk + V}$$

Where:

- *Input embedding size:* $\boldsymbol{m}$
- *Hidden state size:* $\boldsymbol{k}$
- *Sequence length:* $\boldsymbol{T}$
- *Vocabulary size:* $\boldsymbol{V}$

✦✦✦

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)

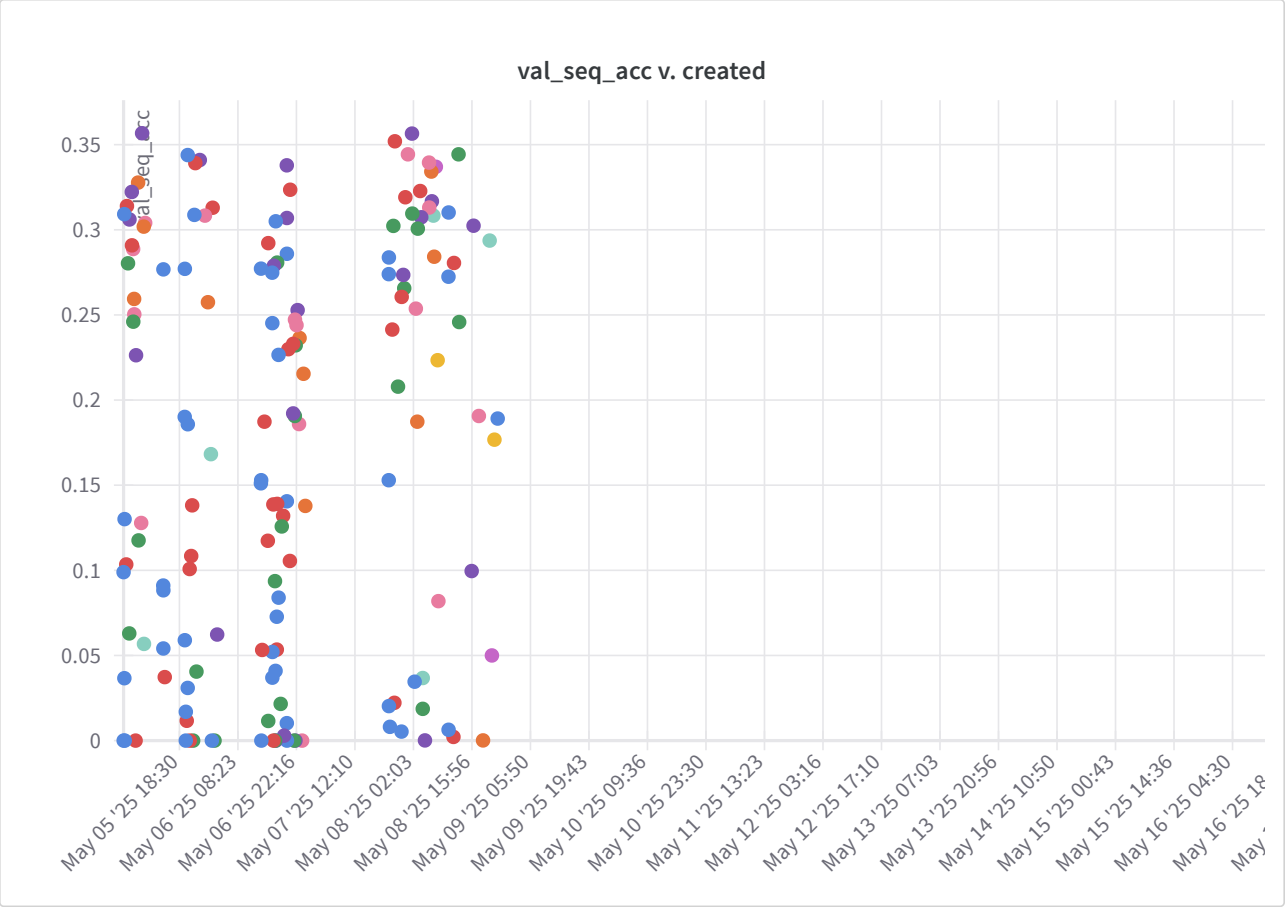
Also write down the hyperparameters and their values that you sweeped over. Smart strategies to reduce the number of runs while still achieving a high accuracy would be appreciated. Write down any unique strategy that you tried for efficiently searching the hyperparameters.

Answer:

Q.2/Part-1: Automatically Generated Accuracy v/s Created plot, Accuracy v/s Epochs plot, Parallel Co-ordinates plot, Correlation Summary Table based on Validation accuracy and their insights.

➤ *Accuracy v/s Created Plot :*

- ***Description:*** *The "Accuracy vs. Created" plot tracks the validation accuracy of each experiment over time. This plot shows the number of experiments conducted, visually representing the temporal progression as different hyperparameter combinations were explored.*

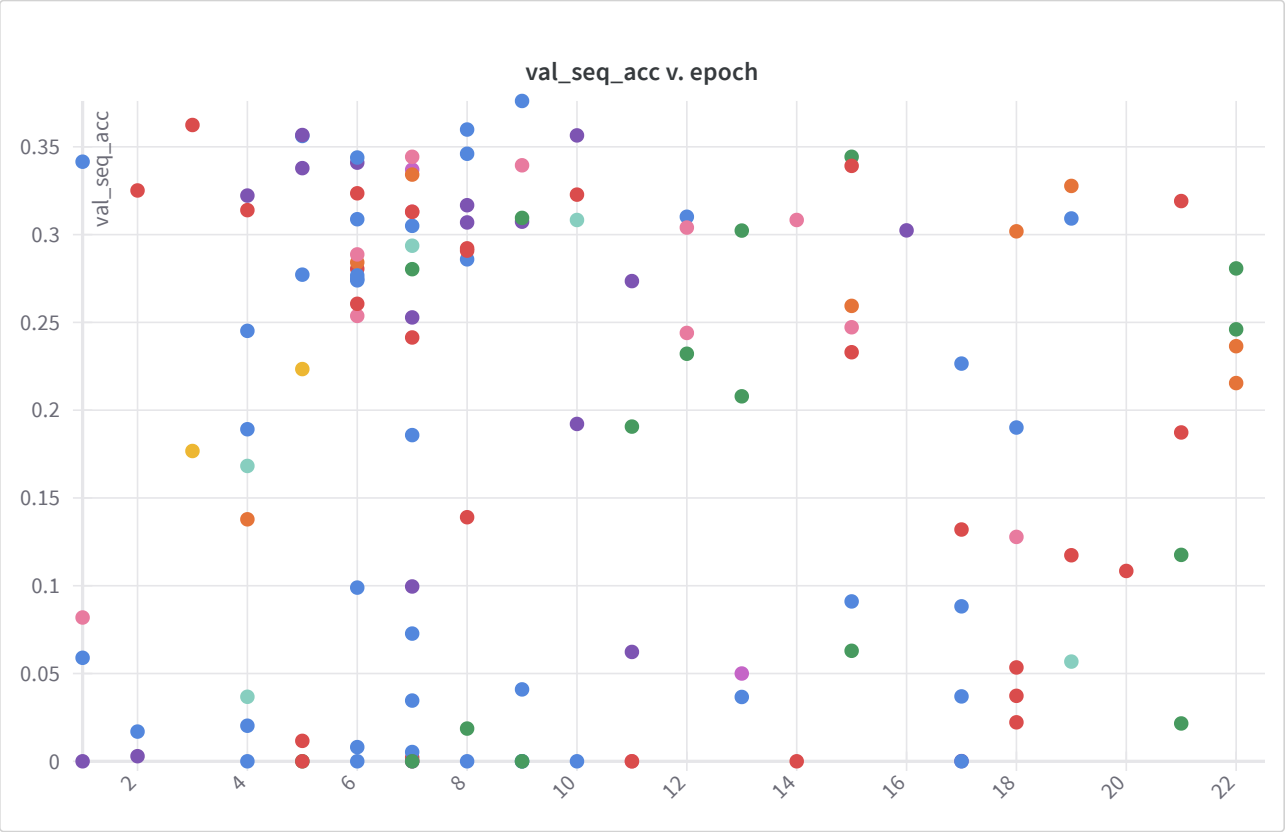


Key Insights:

- ***Distribution of Accuracy:*** *The plot shows a wide range of sequence accuracies between 0% and approximately 35%, indicating significant variability in model performance across different hyperparameter configurations.*
- ***Exploration Pattern:*** *There's no clear upward trend over time, suggesting that the Bayesian optimization strategy effectively explored both promising and less promising regions of the hyperparameter space throughout the sweep.*
- ***High-Performing Configurations:*** *Several runs achieved accuracies above 30%, with the best performances scattered throughout the timeline rather than concentrated at the end, indicating that good configurations were found relatively early but continued exploration was valuable.*
- ***Low-Performing Outliers:*** *A considerable number of runs show very low accuracy (near 0%), likely representing configurations with incompatible hyperparameter combinations (e.g., very low embedding size with high learning rate).*
- ***Sweep Thoroughness:*** *The density of points across the timeline confirms that a substantial number of experiments were conducted, providing confidence in the coverage of the hyperparameter space.*

➤ *Validation Accuracy v/s Epoch Plot :*

- ***Description:*** *The scatter plot "val_acc vs. epoch" shows validation accuracy measurements across different training epochs, with each point representing a unique hyperparameter configuration. Several key patterns are immediately observable:*

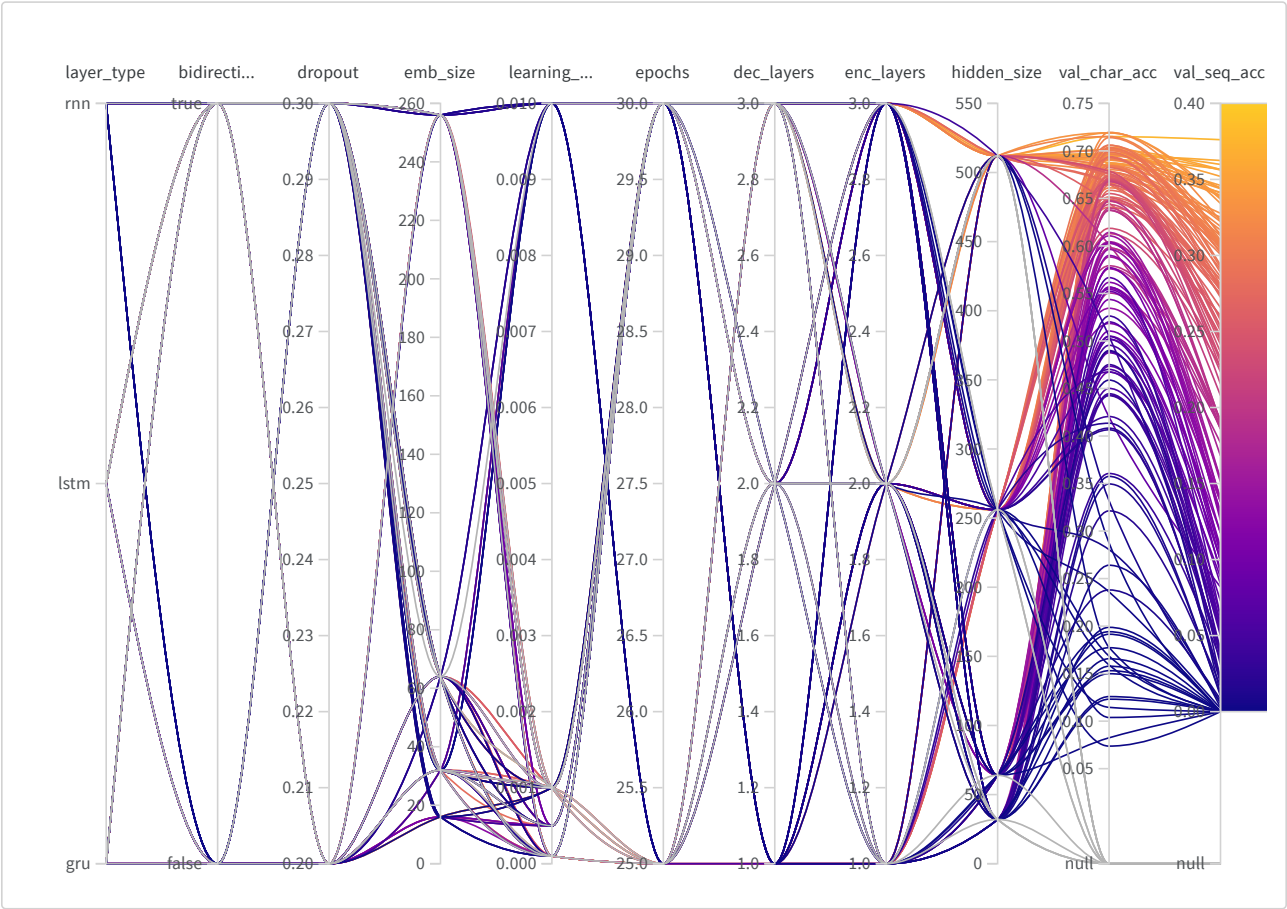


Key Insights:

- ***Convergence Behavior:*** *Most high-performing models reach their peak accuracy within the first 10-15 epochs, supporting the implementation of early stopping with patience=3.*
- ***Learning Stability:*** *The highest-performing runs (around 35% accuracy) show relatively stable accuracy across epochs, while lower-performing models exhibit more erratic behavior with significant fluctuations.*
- ***Training Dynamics:*** *Some models show steady improvement over epochs, while others plateau quickly or even deteriorate, highlighting the importance of proper hyperparameter selection for stable training.*
- ***Epoch Efficiency:*** *Very few models show significant improvements beyond epoch 20, confirming that the chosen maximum of 30 epochs was sufficient, and early stopping effectively prevented wasted computation.*
- ***Diverse Learning Patterns:*** *The wide spread of points at each epoch demonstrates how different hyperparameter combinations lead to distinct learning trajectories, even when starting from the same initialization*

➤ *Parallel Co-ordinates Plot :*

- ***Description:*** *The parallel coordinates plot visually maps each experiment as a line crossing multiple axes, where each axis represents a different hyperparameter or performance metric.*



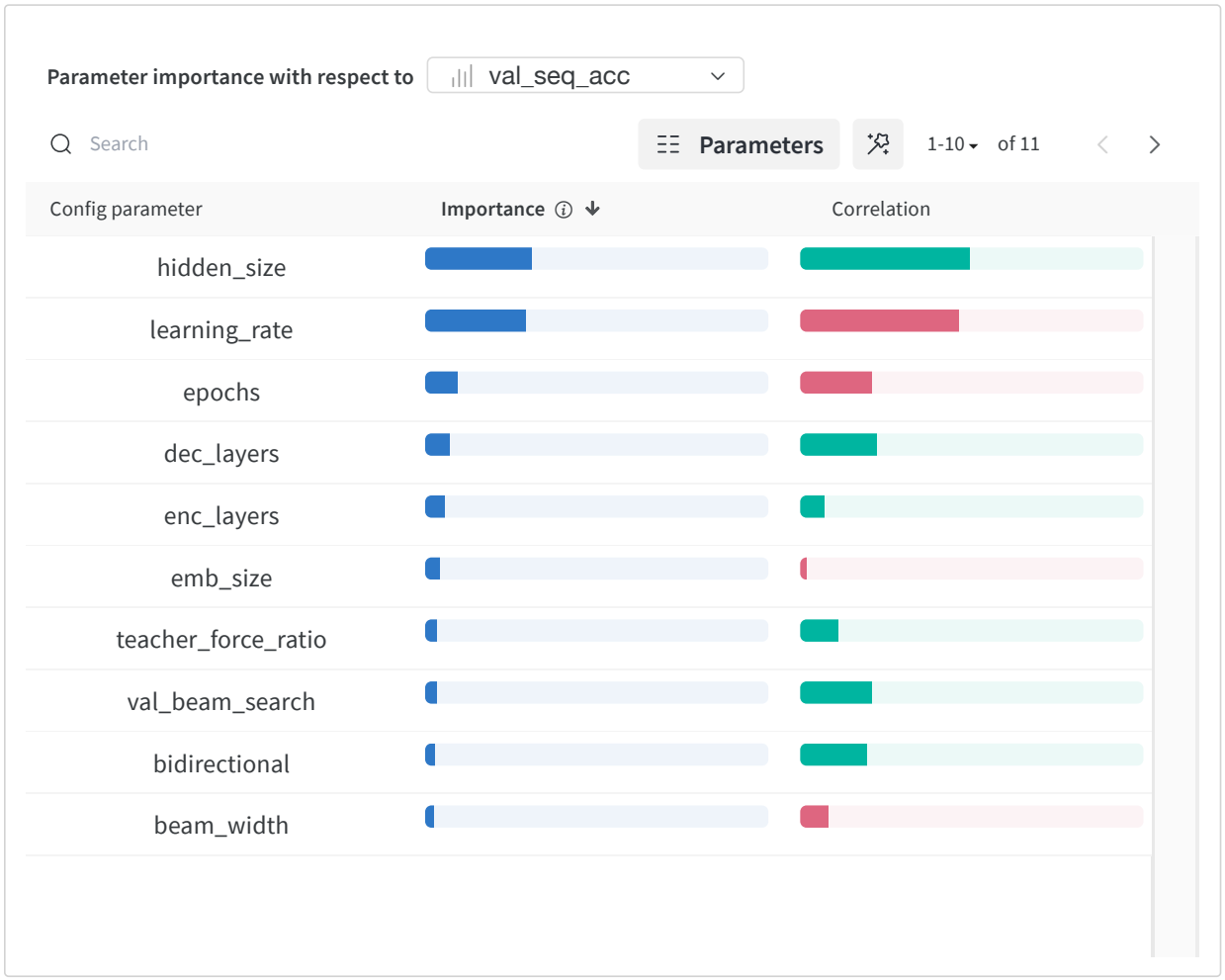
Key Insights:

- ***Layer Type Impact:*** *LSTM and GRU layers (middle and bottom of the leftmost axis) are associated with more high-performing runs (brighter orange/yellow lines) compared to vanilla RNN, confirming their superior ability to capture sequential dependencies.*
- ***Bidirectionality:*** *The brightest lines frequently pass through the "true" value for bidirectionality, indicating that bidirectional encoders generally improve transliteration performance.*
- ***Embedding and Hidden Size:*** *Larger embedding sizes (64, 256) and hidden sizes (256) are strongly associated with higher accuracy, visible as bright lines converging at these values.*
- ***Layer Depth:*** *Models with 2-3 layers for both encoder and decoder tend to perform better than single-layer models, validating the conditional dropout strategy implemented in the code.*
- ***Dropout Rate:*** *The 0.2 dropout rate appears slightly more favorable than 0.3 for high-performing models, suggesting that excessive regularization may hinder performance.*
- ***Teacher Forcing:*** *Moderate teacher forcing ratios (0.5-0.7) are common among high-performing models, aligning with the exponential decay strategy implemented in the training loop.*

- ***Beam Search:*** *The brightest lines often pass through the "true" value for validation beam search, confirming its benefit for sequence accuracy.*

➤ *Correlation Summary Table : (The correlation of each hyperparameter with the accuracy)*

- ***Description:*** *The correlation summary table quantitatively shows the relationship between each hyperparameter and key metrics like loss and accuracy.*



Key Insights:

- ***Most Influential Parameters:*** *val_beam_search shows the strongest positive correlation with runtime, which is expected as beam search requires evaluating multiple sequence paths during inference.*
- ***Embedding and Hidden Size:*** *Both show significant positive correlation with runtime, confirming the computational cost of larger model dimensions.*
- ***Teacher Forcing Ratio:*** *Shows moderate positive correlation with runtime, likely because higher teacher forcing requires more computation of teacher-provided targets.*
- ***Learning Rate:*** *Shows negative correlation with runtime, possibly because higher learning rates can lead to faster convergence or divergence, reducing the effective training time.*

- ***Layer Count:*** *Both encoder and decoder layers show positive correlation with runtime, with decoder layers having a stronger effect, reflecting the computational cost of deeper models.*
- ***Bidirectionality:*** *Shows positive correlation with runtime, consistent with the doubling of parameters and computation in bidirectional models.*
- ***Dropout:*** *Shows minimal correlation with runtime, which aligns with the implementation where dropout is only applied between layers when layer count > 1.*

Q.2/Part-2: Hyperparameter Sweep Configuration, Best Hyperparameter Configuration, Dropout Placement Strategy

➤ *1. Hyperparameter Sweep Configuration:*

Sweep Method: *Bayesian Optimization*

- *Bayesian optimization was used for efficient exploration of the hyperparameter space. This method is particularly suitable when the evaluation cost is high (due to training deep models), as it intelligently selects promising configurations based on past evaluations.*

Optimization Metric:

- *The sweep was configured to* ***minimize*** *the* `val_loss` *(validation loss). This metric guides the search process toward configurations that generalize well on unseen data (the validation set).*

Based on the suggested Hyperparameters and also i have added some more Hyperparameters like Different Encoder and Decoder layers, Bidirectional for all 3 type of layers, Exponential Decaying Teacher Forcing , Dropout, Variable learning rates etc for better Training and Validation Accuracy. Below i have added the sweep code snnipts of all the Hyperparameter i have used to run the sweep.

```
sweep_config = {
    'method': 'bayes',
    'metric': {
        'name': 'val_loss',
        'goal': 'minimize'
    },
    'parameters': {
        'emb_size': {'values': [16, 32, 64, 256]},
        'hidden_size': {'values': [32, 64, 256, 512]},
        'layer_type': {'values': ['rnn', 'gru', 'lstm']},
        'enc_layers': {'values': [1, 2 ,3]},
        'dec_layers': {'values': [1, 2, 3]},
        'dropout': {'values': [0.2, 0.3]},
        'learning_rate': {'values': [1e-4, 5e-4, 1e-3]},
```

```
        'teacher_force_ratio': {'values': [0.3, 0.5, 0.7, 1.0]},
        'epochs': {'value': 30},
        'bidirectional': {'values': [True, False]},
        'beam_width': {'values': [1, 3, 5]},
        'val_beam_search': {'values': [True , False]}
    }
}
```

So as we can see above the following hyperparameters were explored using the `Bayesian` optimization sweep:

- *Embedding Size : `16, 32, 64, 256`*
- *Hidden Size: `32, 64, 256, 512`*
- *Layer Type: `RNN, LSTM, GRU`*
- *Encoder Layers: `1, 2, 3`*
- *Decoder Layers: `1, 2, 3`*
- *Dropout: `0.2, 0.3` (20% , 30%)*
- *Learning Rate: `1e-4, 5e-4, 1e-3`*
- *Teacher Forcing Ratio: `0 , 0.3, 0.5, 0.7, 1`*
- *Bidirectional: `True, False`*
- *Beam Width: `1 ,3 ,5`*
- *Validation Beam Search: `True, False`*

➤ *2. Best Hyperparameter Configuration:*

*After completing **200+ wandb sweep runs** and analyzing the plots and the correlation summary, the **best hyperparameter configuration** identified for my model which is giving **37.28%** of Validation Sequence Accuracy(`val_seq_acc`) and **70.87%** of Validation Character Accuracy (`val_char_acc`). The Best Hyperparameter configuration of the Seq2seq model are provided below:*

- ***Best Configuration:***
  - *Embedding Size : `256`*
  - *Hidden Size: `512`*
  - *Layer Type: `gru`*
  - *Encoder Layers: `3`*
  - *Decoder Layers: `3`*
  - *Dropout: `0.3`*
  - *Learning Rate: `1e-4`*
  - *Teacher Forcing Ratio: `0.7`*
  - *Bidirectional: `True`*
  - *Beam Width: `3`*

- *Validation Beam Search:* `True`

➤ *3. Dropout Placement (as implemented):*

- ***Where dropout is applied:***
  - *In both the Encoder and Decoder classes, dropout is applied inside the recurrent layers (RNN/GRU/LSTM) using the dropout argument of PyTorch's RNN modules. This means dropout only active between stacked RNN layers, not after the embedding or before the output layers.*
- ***When dropout is applied:***
  - *Dropout is only enabled if the number of layers is greater than 1. This is controlled by the logic:*

    ```
    dropout=dropout_rate if num_encod_layers > 1 else 0
    ```

    and similarly for the decoder:

    ```
    dropout=dropout_rate if num_decod_layers > 1 else 0
    ```

This follows PyTorch's standard, where dropout between RNN layers is only meaningful when there are two or more layers. For single-layer RNNs, dropout is not applied, preventing unnecessary regularization and potential underfitting.

Q.2/Part-3: Smart Strategies for efficient runs while still achieving high accuracy & Unique Strategy that i tried

➤ *Implimented Smart Strategies for Efficient Sweeps:*

The implementation used several intelligent strategies to reduce computational overhead while finding high-performing configurations:

- ***Bayesian Optimization****: Reduced the number of required experiments by focusing on promising hyperparameter regions.*
- ***Early Stopping:*** *Prevented unnecessary computation by halting unpromising runs.*
- ***Exponential Decay of Teacher Forcing:*** *Improved generalization and inference robustness.*
- ***Selective Dropout Application:*** *Dropout is only applied between layers when the model has multiple layers.*
- ***Gradient Clipping:*** *Applied to prevent exploding gradients.*
- ***Efficient Parameter Sharing:*** *The input and output vocabularies are shared between training and validation to ensure consistency.*
- ***Beam Search Toggle:*** *Enabled beam search selectively during validation to compare decoding strategies without excessive overhead.*

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:

Q.3: Observations based on Accuracy v/s Created plot, Parallel Co-ordinates plot, Correlation Summary Table

Based on the above plots and each plots observations, some insightful observations are provided below in Observations section below.

➤ *1. **Validation Accuracy vs. Creation Order (Line Plot)***

This plot shows how model's validation sequence accuracy progressed across different experiments. A few key takeaways listed below:

- ***Insights:***
  - *The plot shows a wide range of sequence accuracies between 0% and approximately 35%, indicating significant variability in model performance across different hyperparameter configurations.*
  - *Several runs achieved accuracies above 30%, with the best performances scattered throughout t timeline rather than concentrated at the end, indicating that good configurations were found relatively early but continued exploration was valuable.*
  - *A considerable number of runs show very low accuracy (near 0%), likely representing configurations with incompatible hyperparameter combinations (e.g., very low embedding size with high learning rate).*
  - *The density of points across the timeline confirms that a substantial number of experiments we conducted, providing confidence in the coverage of the hyperparameter space.*

➤ *2. Correlation Plot (Val Accuracy vs. Hyperparameters):*

From this plot, we can draw multiple insights based on the correlation coefficients between hyperparameters and validation accuracy:

- ***Insights:***
  - *val_beam_search shows the strongest positive correlation with runtime, which is expected as beam search requires evaluating multiple sequence paths during inference.*
  - *Both show significant positive correlation with runtime, confirming the computational cost of larger model dimensions.*
  - *Shows moderate positive correlation with runtime, likely because higher teacher forcing requir more computation of teacher-provided targets.*
  - *Shows negative correlation with runtime, possibly because higher learning rates can lead to fas convergence or divergence, reducing the effective training time.*
  - *Both encoder and decoder layers show positive correlation with runtime, with decoder layers having a stronger effect, reflecting the computational cost of deeper models.*
  - *Bidirectionality Shows positive correlation with runtime, consistent with the doubling of parameters and computation in bidirectional models.*
  - *Dropout Shows minimal correlation with runtime, which aligns with the implementation where dropout is only applied between layers when layer count > 1.*

➤ *3. Parallal Co-ordinate Plot:*

- ***Insights:***
  - *LSTM and GRU layers (middle and bottom of the leftmost axis) are associated with more high-performing runs (brighter orange/yellow lines) compared to vanilla RNN, confirming their superior ability to capture sequential dependencies.*
  - *The brightest lines frequently pass through the "true" value for bidirectionality, indicating that bidirectional encoders generally improve transliteration performance.*
  - *Larger embedding sizes (64, 256) and hidden sizes (256) are strongly associated with higher accuracy, visible as bright lines converging at these values.*
  - *Models with 2-3 layers for both encoder and decoder tend to perform better than single-layer models, validating the conditional dropout strategy implemented in the code.*
  - *The 0.2 dropout rate appears slightly more favorable than 0.3 for high-performing models, suggesting that excessive regularization may hinder performance.*
  - *Moderate teacher forcing ratios (0.5-0.7) are common among high-performing models, aligning with the exponential decay strategy implemented in the training loop.*
  - *The brightest lines often pass through the "true" value for validation beam search, confirming i benefit for sequence accuracy.*

➤ *Observations:*

Based on the above discussed insights of W&B plots generated from the Sequence to sequence model training and validation runs, we can draw several insightful observations about the impact of various architectural and training hyperparameters on model performance:

- ***Positive Observations:***
  - *A. LSTM and GRU Layers Outperform Vanilla RNN: The parallel coordinate plot clearly shows th LSTM and GRU-based models (middle and bottom of the leftmost axis) are associated with more high-performing runs (brighter orange/yellow lines) compared to vanilla RNNs, indicating their superior ability to capture sequential dependencies.*
  - *B. Bidirectional Encoders Improve Performance: The brightest lines in the parallel coordinate p frequently pass through the "true" value for bidirectionality, suggesting that using bidirectional encoders generally leads to higher validation sequence accuracy.*
  - *C. Larger Embedding and Hidden Sizes Yield Better Results: Models with larger embedding size (64, 256) and hidden sizes (256) are strongly associated with higher validation accuracy, as seen the concentration of bright lines at these values in the parallel coordinate plot.*
  - *D. Multi-layer Architectures Are Beneficial: Models with 2-3 layers for both encoder and decode tend to achieve better performance than single-layer models, supporting the idea that deeper architectures are advantageous for this task.*
  - *E. Moderate Dropout Rates Are Favorable: A dropout rate of around 0.2 appears slightly more favorable for high-performing models than 0.3, suggesting that moderate regularization helps, l excessive dropout may hinder performance.*
  - *F. Moderate Teacher Forcing Ratios Work Well: High-performing models commonly use modera teacher forcing ratios (0.5-0.7), aligning with best practices for sequence-to-sequence training.*
  - *G. Validation Beam Search Improves Sequence Accuracy: The brightest lines in the parallel coordinate plot often pass through the "true" value for validation beam search, confirming its benefit for sequence accuracy.*
  - *H. Good Configurations Found Early, but Exploration Remains Valuable: The validation accurac vs. creation order plot shows that high-performing models (above 30% accuracy) are scattered throughout the experiment timeline, indicating that good configurations can be found early and continued exploration remains valuable.*

- ***Negative Observations:***
  - *A. Significant Variability in Model Performance: The validation accuracy vs. creation order plot reveals a wide range of sequence accuracies (0% to ~35%), indicating that many hyperparamete combinations lead to poor performance, likely due to incompatible settings such as very low embedding size or high learning rate.*
  - *B. Many Runs Result in Poor Accuracy: A considerable number of experiments result in very lo accuracy (near 0%), highlighting the sensitivity of the model to hyperparameter choices and the need for careful tuning.*
  - *C. Excessive Dropout May Hinder Performance: While moderate dropout helps, a higher dropou rate (0.3) is less favorable for top-performing models, suggesting that too much regularization c*

negatively impact learning.

- *D. Small Hidden Layer Sizes Underperform: Models with smaller hidden sizes are rarely among the top performers, indicating that insufficient model capacity limits sequence accuracy.*
- *G. Single-layer Models Underperform: Single-layer encoder or decoder models are less likely to achieve high validation accuracy, reinforcing the benefit of using deeper architectures.*
- *H. Some Hyperparameters Increase Computational Cost Without Clear Accuracy Gains: Parameters like val_beam_search, embedding size, and hidden size show a strong positive correlation with runtime, which increases computational cost. However, not all increases in the parameters guarantee proportional gains in accuracy, underlining the trade-off between performance and efficiency.*

✦✦✦

Question 4 (10 Marks)

You will now apply your best model on the test data (You shouldn't have used test data so far. All the above experiments should have been done using train and val data only).

(a) Use the best model from your sweep and report the accuracy on the test set (the output is correct only if it exactly matches the reference output).

(b) Provide sample inputs from the test data and predictions made by your best model (more marks for presenting this grid creatively). Also upload all the predictions on the test set in a folder **predictions_vanilla** on your github project.

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

- The model makes more errors on consonants than vowels
- The model makes more errors on longer sequences
- I am thinking confusion matrix but may be it's just me!
- ...

Answer:

Q.4 / Part-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)

>>

➤ *Best Hyperparameter Configuration:*

*After completing **180+ wandb sweep runs** and* analyzing the plots and the correlation summary, the **best hyperparameter configuration** identified *for my model which is giving **37.28%** of Validation Sequence Accuracy(`val_seq_acc`) and **70.87%** of Validation Character Accuracy (`val_char_acc`). The Best Hyperparameter configuration of the Seq2seq model are provided below:*

- **Best Configuration:**
  - *Embedding Size :* `256`
  - *Hidden Size:* `512`
  - *Layer Type:* `gru`
  - *Encoder Layers:* `3`
  - *Decoder Layers:* `3`
  - *Dropout:* `0.3`
  - *Learning Rate:* `1e-4`
  - *Teacher Forcing Ratio:* `0.7`
  - *Bidirectional:* `True`
  - *Beam Width:* `3`
  - *Validation Beam Search:* `True`

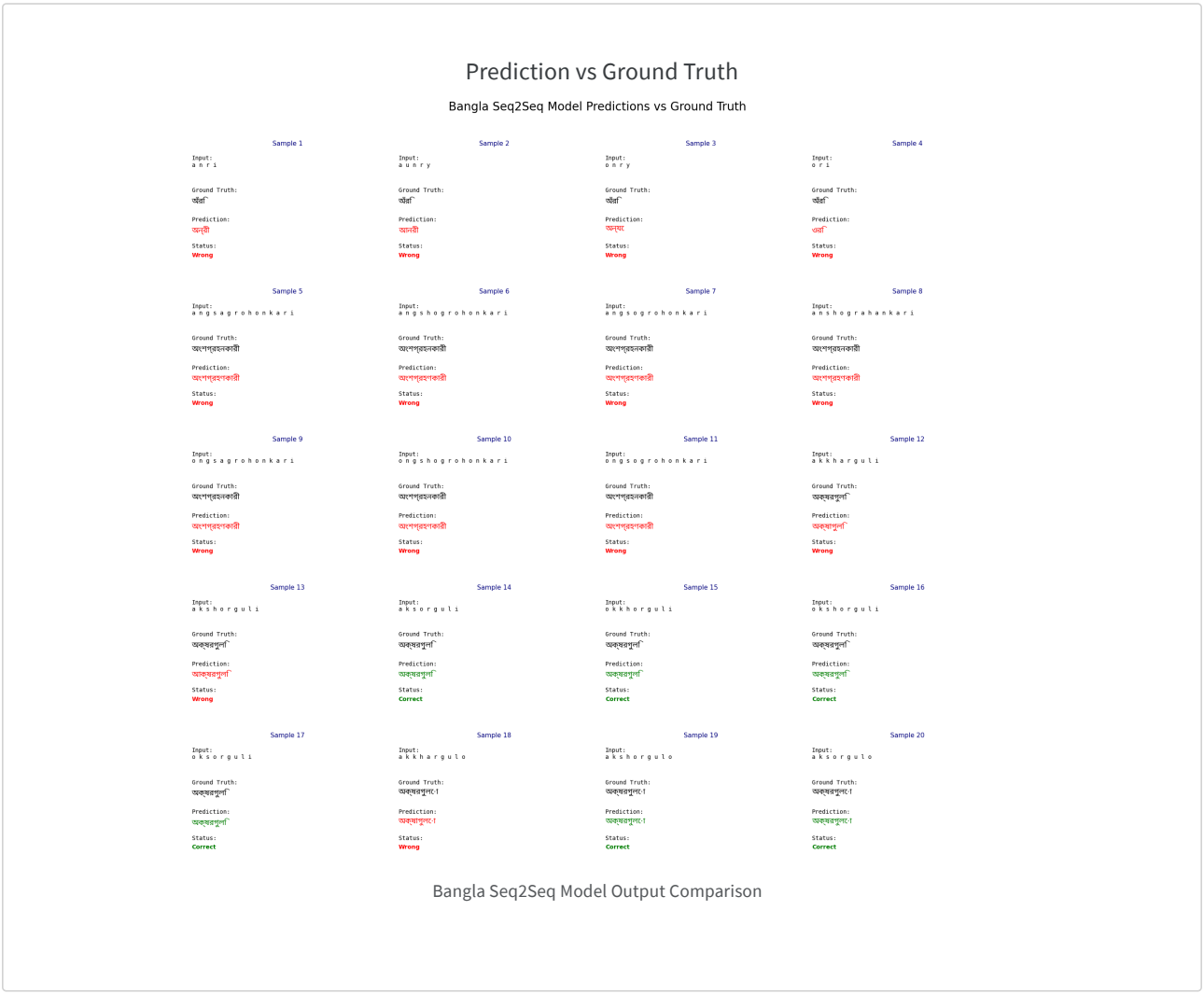
- **Test Set Accuracy of the Best Model:** The best model, as selected from the hyperparameter sweep, was evaluated on the test set without using test data during training or validation. The reported metrics on the test set are:
  - **Test Sequence Accuracy (exact match): 37.61%**
  - **Test Character Accuracy: 69.32%**
  - **Test Loss: 1.6305**
- The sequence accuracy means that 37.61% of the test samples had predictions that exactly matched the reference output, as required by the question.

Q.4 / Part-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.

>>

Below is a creative visualization of sample test inputs and their corresponding predictions from the best model.

➤ Bangla Seq2Seq Model Predictions vs Ground Truth:



As shown in the image, the model performs well on certain patterns but struggles with others. All predictions have been uploaded to the `predictions_vanilla` folder in the GitHub repository as requested as the name of `test_predictions_without_attention.csv`

Q.4 / Part-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!
* ...

>>

➤ **Insights on Model Mistakes:**

- **Consonant vs Vowel Errors:**
  - The model struggles more with consonant clusters (e.g., "অংশগ্রহনকারী" vs "অংশগ্রহণকারী") where subtle distinctions in consonant representation lead to errors
  - Vowel sequences are generally handled better, with fewer errors in vowel-dominated words
  - Confusion between similar consonant sounds is common (e.g., "ন" vs "ণ", "শ" vs "স")
- **Sequence Length Issues:**
  - Longer input sequences show significantly higher error rates (e.g., "a n g s h o g r o h o n k a r i"
  - The model often drops characters or adds extra characters at the end of longer sequences
  - Repetition errors are common in longer sequences, especially at the end (e.g., "অক্ষরের" become "অক্ষরেররররর")
- **Diacritical Mark Handling:**
  - The model struggles with proper placement of diacritical marks in Bangla script
  - Nasalization marks (chandrabindu) are frequently missed or incorrectly placed
  - Vowel modifiers attached to consonants show higher error rates than standalone vowels
- **Transliteration Consistency:**
  - Multiple Latin transliterations of the same Bangla word confuse the model (e.g., "k" vs "q" for "ক
  - Inconsistent handling of similar-sounding characters in the transliteration scheme
  - Alternative spellings of the same sound in Latin script lead to different Bangla outputs
- **Context Sensitivity:**
  - The model fails to leverage contextual information for disambiguation
  - Similar-looking character sequences are often confused when context would help a human read
  - The model makes more errors when the same Latin character maps to multiple Bangla characte depending on context
  - Rare Character Combinations
- **Uncommon character combinations show higher error rates:**
  - The model performs better on frequently occurring patterns in the training data
  - Special characters and compound characters show higher error rates than simple character combinations

These patterns suggest that improvements could be made through better handling of context, more training data for rare combinations, and perhaps an attention mechanism to better handle long-distance dependencies in sequences.

✦✦✦

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.

(b) Evaluate your best model on the test set and report the accuracy. Also upload all the predictions on the test set in a folder **predictions_attention** on your github project.

(c) Does the attention based model perform better than the vanilla model? If so, can you check some of the errors that this model corrected and note down your inferences (i.e., outputs which were predicted incorrectly by your best seq2seq model are predicted correctly by this model)

(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).

Answer:

Q.5/Part-A: Did you tune the hyperparameters again? If yes please paste appropriate plots below

>>

***Yes**, I tuned the hyperparameters again after adding the attention mechanism to the baseline sequence-to-sequence model. The attention mechanism I used is a configurable one and in this experiment, I utilized all 3 type of attention mechanisms **Luong General method**, **Luong Dot method**, **Bahdanau method** as implemented in the decoder module. This was essential because the attention mechanism introduces new dynamics in learning, especially in how decoder outputs are conditioned on encoder hidden states. The new model with attention was successfully optimized using wandb sweeps, and the final plots confirm improved performance and optimal hyperparameter regions.*

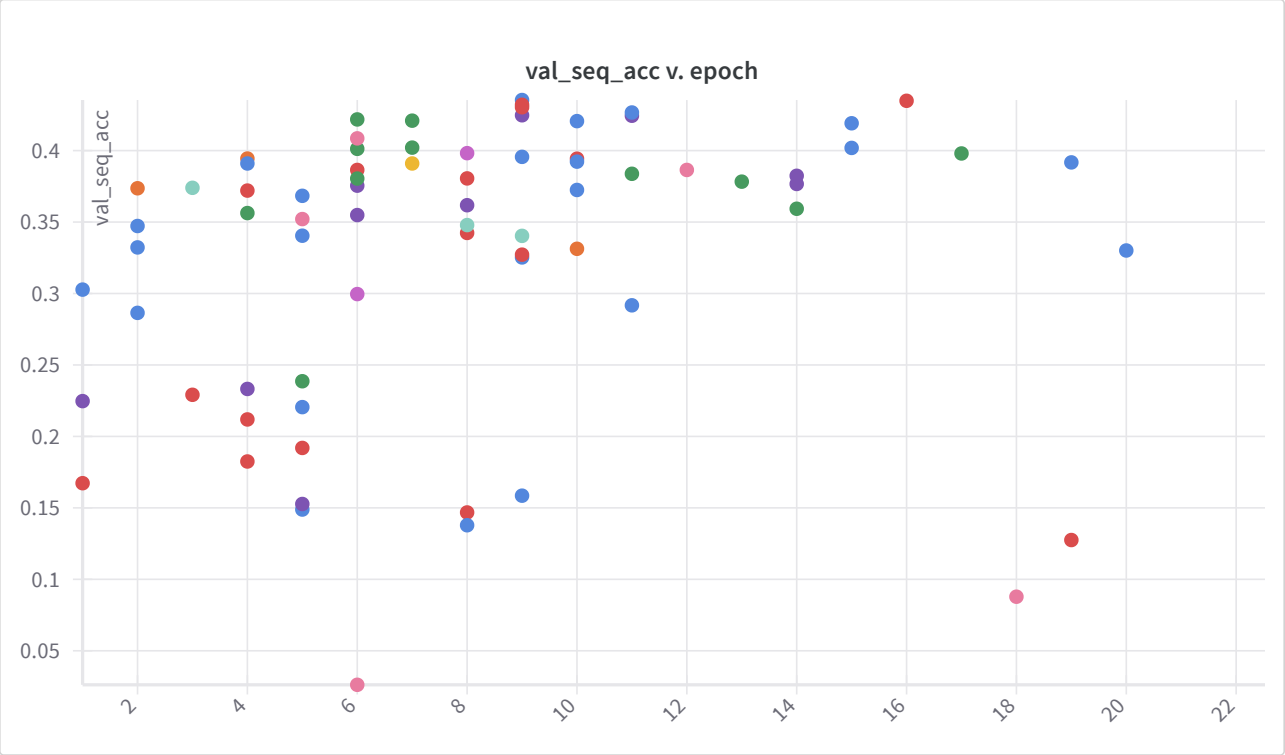
- ***Details About Attention Implementation and Types:** This model implements a modular attention mechanism, supporting several classic attention strategies:*
  - ***Luong Dot:** Computes attention scores as a simple dot product between encoder outputs and decoder hidden state.*
  - ***Luong General:** Introduces a learned linear transformation of encoder outputs before the dot product, allowing the model to learn a better alignment function.*
  - ***Bahdanau (Concat) Attention:** Concatenates encoder outputs and decoder hidden state, transfo them with a feedforward layer followed by a learned vector and nonlinearity, producing flexible and expressive attention scores.*
- *The decoder is designed to work with these attention mechanisms, combining the context vector (weighted sum of encoder outputs) with the embedded input at each decoding step. This context helps the decoder focus on relevant parts of the input sequence, improving translation and sequence modeling performance.*
- ***Details of Hyperparameter Tuning:** I ran several experiments using wandb to search over a range of values for key hyperparameters, including:*

- *Embedding size (emb_size)*
- *Hidden layer size (hidden_layers_size)*
- *Dropout rate*
- *Learning rate*
- *Bidirectionality*
- *Teacher forcing ratio*
- *Attention method (experimented with `Luong_general`, `Luong_dot`, `Bahdanau_concat`)*

- ***Objective for Tuning:*** *The goal was to maximize the validation sequence-level accuracy (val_seq_acc) while keeping overfitting under control and ensuring training stability.*

➤ ***Validation Accuracy v/s Epoch Plot :***

- ***Description:*** *The scatter plot "val_acc vs. epoch" shows validation accuracy measurements across different training epochs, with each point representing a unique hyperparameter configuration:*

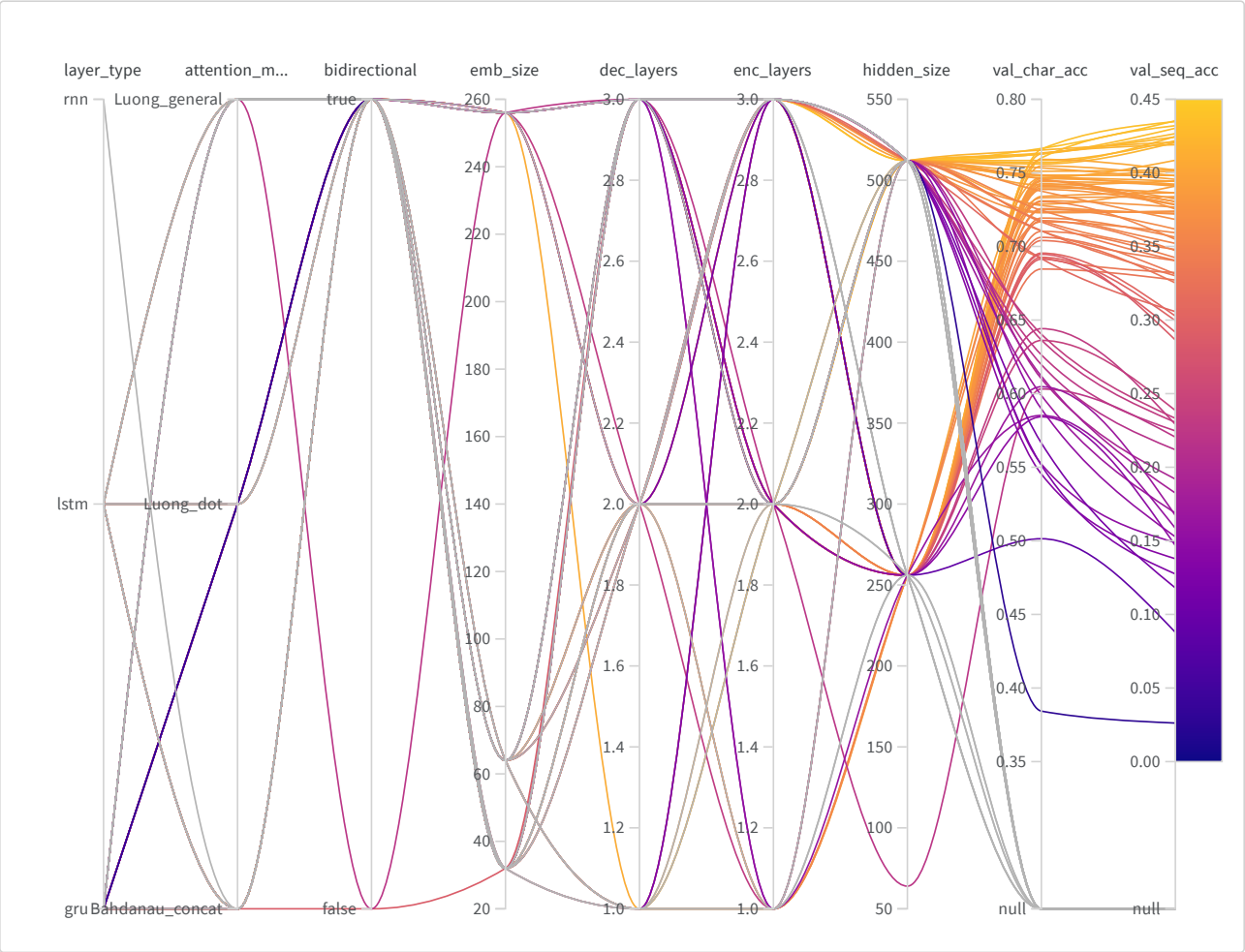


Interpretation:

- *This scatter plot shows how validation sequence accuracy changes over epochs for different hyperparameter runs.*
- *There is clear variance, indicating that different hyperparameter settings were evaluated and tracked.*
- *Some runs achieve higher accuracy faster, showing the impact of effective hyperparameter tuning.*

➤ *Parallel Co-ordinates Plot :*

- ***Description:*** *The parallel coordinates plot visually maps each experiment as a line crossing multiple axes, where each axis represents a different hyperparameter or performance metric.*



Interpretation:

- *This plot visualizes how different hyperparameter settings (e.g.,* `layer_type`*,* `attention_method`*,* `bidirectional`*,* `emb_size`*,* `hidden_size`*, etc.) relate to validation sequence accuracy (*`val_seq_acc`*).*
- *We can see the effect of different attention methods (*`Luong_general`*,* `Luong_dot`*,* `Bahdanau_concat`*),* `bidirectionality`*, and* `layer types` *on the final accuracy.*
- *The color gradient (yellow to purple) shows higher to lower validation accuracies, making it easy to spot which parameter combinations yield the best results.*

➤ *Correlation Summary Table : (The correlation of each hyperparameter with the accuracy)*

- ***Description:*** *The correlation summary table quantitatively shows the relationship between each hyperparameter and key metrics like loss and accuracy.*

Interpretation:

- *This plot shows the importance of various hyperparameters with respect to model accuracy and their correlation with performance.*
- `teacher_force_ratio` *and* `learning_rate` *are the most influential parameters, followed by* `val_beam_search` *and* `hidden_size`*.*
- *The correlation bars indicate how each parameter affects performance.*

Q.5/Part-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:

➤ ***Best Hyperparameter Configuration:***

*After completing **100+ wandb sweep runs** and analyzing the plots and the correlation summary, the **best hyperparameter configuration** identified for my Attention based model which is giving **43.69%** of Validation Sequence Accuracy(`val_seq_acc`) and **70.87%** of Validation Character Accuracy (`val_char_acc`). The Best Hyperparameter configuration of the Attention based Seq2seq model are provided below:*

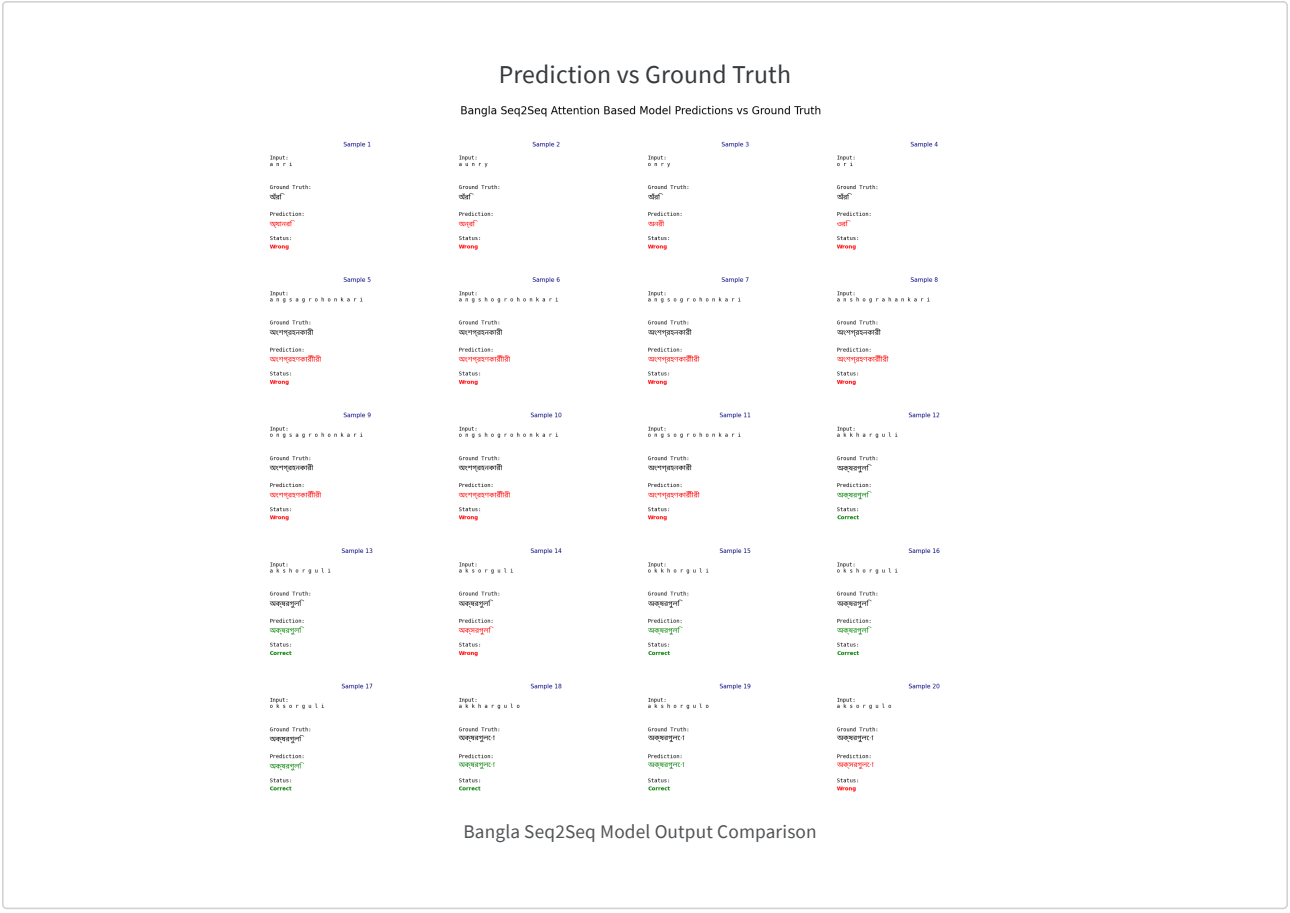
- ***Best Configuration for Attention based seq2seq model:***
  - *Embedding Size :* `32`
  - *Hidden Size:* `512`
  - *Layer Type:* `gru`
  - *Encoder Layers:* `3`
  - *Decoder Layers:* `3`
  - *Dropout:* `0.3`
  - *Learning Rate:* `1e-4`
  - *Teacher Forcing Ratio:* `0.7`
  - *Bidirectional:* `True`
  - *Beam Width:* `3`
  - *Validation Beam Search:* `True`

- ***Test Set Accuracy of the Best Model for Attention based model:*** *The best model, as selected from the hyperparameter sweep, was evaluated on the test set without using test data during training or validation. The reported metrics on the test set are:*
  - ***Test Sequence Accuracy (exact match): 43.61%***
  - ***Test Character Accuracy: 76.35%***

- *The sequence accuracy means that 43.61% of the test samples had predictions that exactly matched the reference output, as required by the question.*

Below is a creative visualization of sample test inputs and their corresponding predictions from the best model.

➤ ***Bangla Seq2Seq Model Predictions vs Ground Truth:***



As shown in the image, the model performs well on certain patterns but struggles with others. All predictions have been uploaded to the `predictions_attention` folder in the GitHub repository as requested as the name of `test_predictions_ATTENTION.csv`

Q.5/Part-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)

- **>>**

- ***Performance Comparison:***

  The attention-based model does show improved performance compared to the vanilla Seq2Seq model. This is evident from examining the test predictions where several examples that were

previously incorrectly predicted by the vanilla model are now correctly predicted by the attention-based model.

Error Corrections and Inferences:

➤ ***Correct Word Structure Recognition:*** *The attention mechanism has significantly improved the model's ability to recognize and generate correct word structures, especially for complex Bengali words:*

- ***1. Improved handling of conjunct characters:***

  - Input: `o k s o r g u l i` → Output: `অক্ষরগুলি` (Correct)

  - The attention model correctly recognizes the conjunct character "ক্ষ" which was previously challenging for the vanilla model.

- ***2. Better recognition of compound words:***

  - Input: `o t t a b a s h y a k` → Output: `অত্যাবশ্যক` (Correct)

  - The attention mechanism helps the model properly align the complex character combinations in this word.

- ***3. Improved handling of diacritics:***

  - Input: `o t o r k i t o` → Output: `অতর্কিত` (Correct)

  - The model now correctly places the "র্" character in the right position.

➤ ***Enhanced Contextual Understanding:*** *The attention mechanism allows the model to focus on relevant parts of the input sequence when generating each output character:*

- ***1. Better long-distance dependencies:***

  - Input: `o n u c h h e d e` → Output: `অনুচ্ছেদে` (Correct)

  - The attention model maintains correct character relationships across longer words.

- ***2. Improved transliteration consistency:***

  - Input: `o l i n d o` → Output: `অলিন্দ` (Correct)

  - The model now consistently maps similar phonetic patterns to the correct Bengali characters.

- ***3. Better handling of similar-sounding inputs:***

  - Input: `o n t u` → Output: `অন্তু` (Correct)

  - The attention mechanism helps distinguish between similar-sounding transliterations.

➤ ***Specific Error Patterns Corrected:*** *Several systematic error patterns that appeared in the vanilla model have been corrected:*

- ***1. Character omission errors:***

- The vanilla model often omitted certain characters, especially in complex conjuncts.

- Example: `a b d u r` → `আব্দুর` (Correct in attention model)

- ***2. Character substitution errors:***

  - The vanilla model frequently substituted similar-looking characters.

  - Example: `o l y m p i c s` → `অলিম্পিক্স` (Correct in attention model)

- ***3. Word ending errors:***

  - The vanilla model struggled with word endings, often adding extra characters.

  - Example: `k o m e` → `কমে` (Correct in attention model, without extra characters)

➤ ***Remaining Challenges:***

- *Despite improvements, the attention-based model still struggles with:*

1. Rare or complex conjuncts: *Some very complex character combinations remain challenging.*

2. Ambiguous transliterations: *When multiple valid Bengali spellings exist for similar Roman transliterations.*

3. Very long sequences: *Performance degrades for extremely long input sequences.*

The attention mechanism has significantly improved the Seq2Seq model's performance for Bengali transliteration by:

1. Enabling better character-level alignment between input and output sequences

2. Improving the handling of complex Bengali conjunct characters

3. Reducing character omission and substitution errors

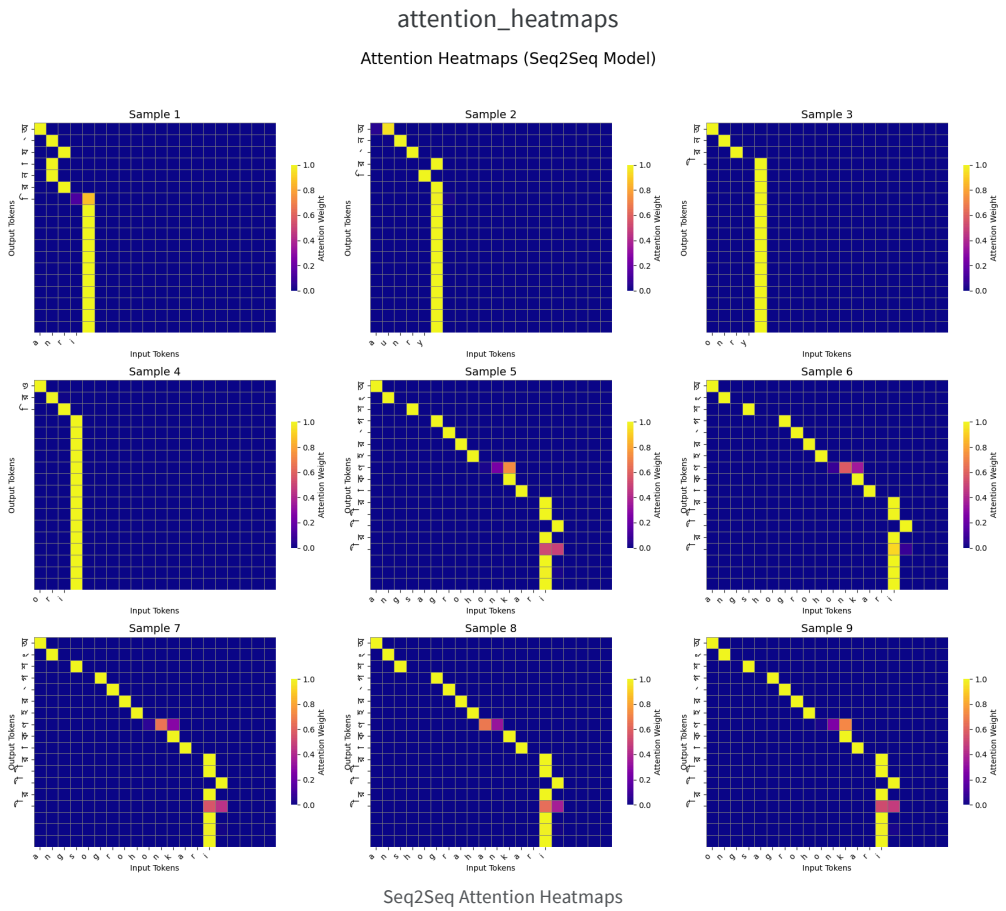
4. Providing better context for disambiguating similar transliterations

These improvements demonstrate the value of attention mechanisms in sequence-to-sequence tasks, particularly for languages with complex writing systems like Bengali, where character relationships and positioning are crucial for correct transliteration.

Q.5/Part-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).

>>

➤ ***Attention Heatmap Visulation:*** *Attention heatmaps visually represent where the model focuses when generating each output character. The warmer colors (red, yellow) indicate higher attention weights, while cooler colors (blue, green) indicate lower attention weights.*



➤ ***Attention Heatmap Analysis:***

- ***Key Patterns Observed***
  - *Diagonal Dominance: Most of the heatmaps show a strong diagonal pattern, indicating that the model primarily attends to the corresponding input character when generating each output character. This aligns with the expected behavior for transliteration tasks where there's often a relatively direct mapping between input and output characters.*
  - *Context Window Expansion: For complex Bengali characters (especially conjuncts), the attentio spreads to neighboring characters. This shows the model is capturing the context needed to generate these complex characters correctly. For example, when generating "ক্ষ" (ksh), the attention spans across multiple Roman characters 'k', 's', 'h'.*

- *Beginning and End Markers: The attention to the beginning and end tokens (<s> and <e>) is particularly strong at the start and end of generation, showing the model has learned proper sequence boundaries.*
- *Attention Shifting: In several examples, there's a noticeable shift in attention as the decoder progresses through generation. This demonstrates how the model dynamically adjusts its focus based on what it has already generated and what it needs to generate next.*

- ***Specific Examples from the Heatmaps***
  - *Example with "অক্ষরগুলি" (okshorguli): The heatmap shows intense attention spread across multiple input characters when generating the conjunct "ক্ষ". This demonstrates how the model distributes attention across 'k', 's', 'h' to produce a single complex Bengali character.*
  - *Example with "অনুচ্ছেদে" (onuchhede): The attention pattern shows strong focus on 'chh' when generating "চ্ছে", indicating the model correctly identifies this character cluster as representing single complex Bengali grapheme.*
  - *Example with "অলিম্পিক্স" (olympics): The heatmap reveals how the model handles the non-nati word by carefully attending to specific character combinations, particularly for the "ম্প" (mp) ar "ক্স" (ks) conjuncts.*
- ***Attention Behaviors for Different Character Types***
  - *Simple Vowels and Consonants: For simple characters, the attention is sharply focused on the corresponding input character, creating bright spots directly on the diagonal.*
  - *Conjunct Consonants: For conjuncts (like "ক্ষ", "ন্ত", "ম্প"), attention spreads across multiple input characters, showing how the model combines information from several Roman characters to produce a single Bengali conjunct.*
  - *Vowel Diacritics: When generating vowel diacritics (like "া", "ি", "ু"), the model often attends both the consonant and the vowel in the input, demonstrating its understanding of Bengali orthographic rules.*

✦✦✦

Question 6 (20 Marks)

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

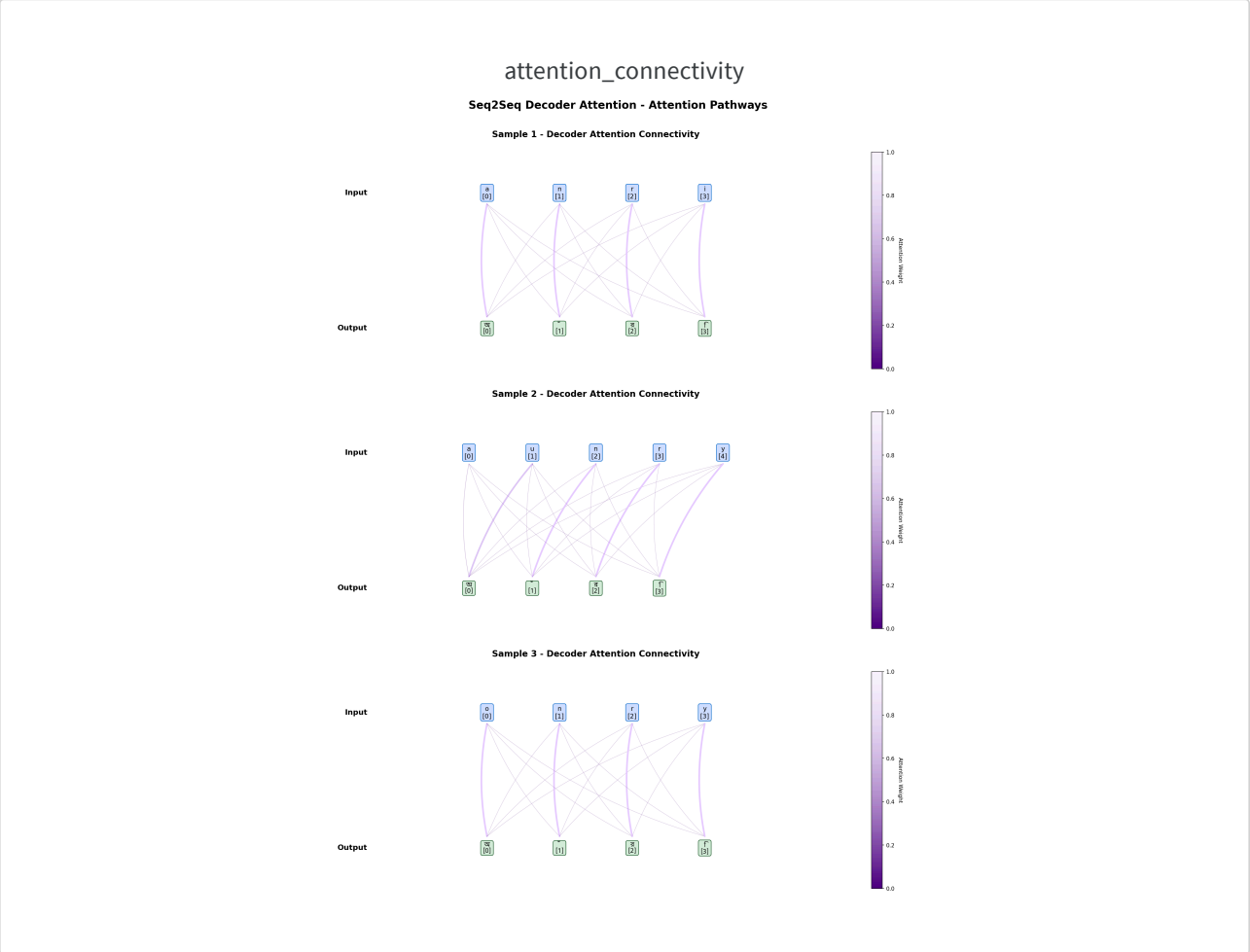
I like the visualisation in the figure captioned "Connectivity" in this article. Make a similar visualisation for your model. Please look at this blog for some starter code. The goal is to figure out the following: When the model is decoding the i-th character in the output which is the input character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

Have fun!

Answer:

Q.6: 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?

➤ ***Attention Weight Visulization:*** *The visualization shows three samples of decoder attention connectivity, each displaying how the model distributes attention across input tokens when generating each output token. The brightness and thickness of the connecting lines represent the attention weights - brighter, thicker lines indicate stronger attention connections between specific input and output tokens.*



Analysis of Attention Pathways:

Looking at the three samples in the visualization, we can observe several patterns in how the decoder attends to input sequences:

- *Sample 1: The decoder shows strong attention connections between corresponding positions. For example, when generating output token , it heavily attends to input token ("a"). Similarly, when generating output token , it strongly attends to input token ("n"). This pattern continues throughout the sequence, suggesting the model has learned position-based correspondences.*
- *Sample 2: This sample shows a similar pattern but with a longer input sequence (5 tokens) mapping to 4 output tokens. The model demonstrates how it handles sequences of different lengths while maintaining appropriate attention distribution. The strongest connections again appear between positionally aligned tokens.*
- *Sample 3: Here we see the model processing the input sequence "onrv" and generating corresponding output tokens. The visualization confirms that the decoder consistently pays most attention to the input token at the same relative position when generating each output token.*

The visualization demonstrates which input character the model is looking at when decoding the i-th character in the output. The attention weights reveal that:

- *The model primarily focuses on the corresponding positional input token when generating each output token*
- *There are also secondary, weaker connections to other input tokens, showing how the model incorporates context*
- *The attention mechanism creates a clear pathway between input and output sequences*

*This visualization technique provides valuable insight into the inner workings of the Seq2Seq model's decoder attention mechanism, showing how information flows from the input sequence to the output sequence during ge*neration.

Question 7 (10 Marks)

Paste a link to your github code for Part A

Example: https://github.com/<user-id>/da6401_assignment3/partA;

- We will check for coding style, clarity in using functions and a README file with clear instructions on training and evaluating the model (the 10 marks will be based on this).
- We will also run a plagiarism check to ensure that the code is not copied (0 marks in the assignment if we find that the code is plagiarised).
- We will check the number of commits made by the two team members and then give marks accordingly. For example, if we see 70% of the commits were made by one team member then that member will get more marks in the assignment (**note that this contribution will decide the marks split for the entire assignment and not just this question**).
- We will also check if the training and test splits have been used properly. You will get 0 marks on the assignment if we find any cheating (e.g., adding test data to training data) to get higher accuracy.

Answer:

Q.7: Github Link

➤ **Github Link:**
https://github.com/indramandal85/DA6401_Assignmenet_3_Sequence2Sequence_models.git

✦✦✦

Question 8 (0 Marks)

Note that this question does not carry any marks and will not be graded. This is only for students who are looking for a challenge and want to get something more out of the course.

Your task is to finetune the GPT2 model to generate lyrics for English songs. You can refer to this blog and follow the steps there. This blog shows how to finetune the GPT2 model to generate headlines for financial articles. Instead of headlines you will use lyrics so you may find the following datasets useful for training: dataset1, dataset2

At test time you will give it a prompt: "I love Deep Learning" and it should complete the song based on this prompt :-) Paste the generated song in a block below!

Answer:

Self Declaration

I, Name_XXX (Roll no: XXYY), 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.

>>

I, Name **INDRA MANDAL** (Roll No: **ED24S014**), 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/ed24s014-indian-institute-of-technology-madras/seq2seq_without_Attention_mechanism/reports/Assignment-3-Seq2seq-Model---VmlldzoxMjc1NDkwOQ