CS6910 - Assignment 3 | CS22M024

Use recurrent neural networks to build a transliteration system.

Avinash Singh Kushwah

For this assignment, I have drawn inspiration from various sources, including:

- Aladdin Pearson's YouTube playlist on Seq2Seq Model, which can be accessed at: YouTube playlist on Seq2Seq Model
- The PyTorch Seq2Seq GitHub repository by bentrevett, available at: GitHub repository
- Insightful "Deep learning" lectures on Guvi delivered by Sir.

Instructions

- The goal of this assignment is threefold: (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
- Discussions with other students is encouraged.
- You must use `Python` for your implementation.
- You can use any and all packages from `PyTorch` or `PyTorch-Lightning`. NO OTHER DL library, such as `TensorFlow` or `Keras` is allowed.
- Please confirm with the TAs before using any new external library. BTW, you may want to explore PyTorch-Lightning as it includes `fp16` mixed-precision training, wandb integration, and many other black boxes eliminating the need for boiler-plate code. Also, do look out for PyTorch2.0.

- You can run the code in a jupyter notebook on Colab/Kaggle 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 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 must also 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 a sample of the Aksharantar dataset released by AI4Bharat. 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 (how 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 realize, this is the problem of mapping a sequence of characters in one language to a sequence of characters in another. 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 a sequence of **characters** here).

Read this blog to understand how to build neural sequence-to-sequence models.

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 mmm, the encoder and decoder have 'l' 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')

Observation:

The total computations done by the network can be calculated as follows:

Encoder Computations:

Calculation for s_1:

Multiplications: $m \cdot k$

Additions: $(m-1) \cdot k$

Total computations: $m \cdot k + (m-1) \cdot k$

Calculations for s_2 to s_t ($t = 2$ to T):

Calculation for $U \cdot x_i$:

Multiplications: $m \cdot k$

Additions: (m-1)·k

Total computations: m·k + (m-1)·k

Calculation for W·s_{i-1}:

Multiplications: k^2

Additions: (k-1)·k

Total computations: k^2 + (k-1)·k

Calculation for U·x_i + W·s_{i-1} + b:

Additions: 2k

Total computations: 2k

Total computations per time-step: $2mk + k^2 + (m-1) \cdot k + (k-1) \cdot k + 2k$

Total computations for T time-steps:$T(2mk + k^2 + (m-1) \cdot k + (k-1) \cdot k + 2k)$

Decoder Computations:

Calculation for softmax(As_i + c):

Multiplication = $O(km + k^2)$

Addition = $O(km + k^2)$

Total computations per time-step: $O(mk + k^2 + mV + Vk + 2V + 2k)$

Total computations for T time-steps: $O(T(mk + k^2 + mV + Vk + 2V + 2k))$

Total computations in the network: Encoder Computations + Decoder Computations

Note: The computation undergoes a relu activation function, which could vary for a sigmoid activation function due to the need for exponentiation.

(b) What is the total number of parameters in your network? (assume that the input embedding size is mmm, the encoder and decoder have 'l' 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')

Observation : Encoders we have : $E \in R(m \times V), U \in R(m \times k), W \in R(k \times k), b \in R(k \times 1)$

To determine the total number of parameters at the encoder side, we can calculate the number of elements in each variable and add them together. The matrix E has m x V elements, the matrix U has m x k elements, the matrix W has k x k elements, and the vector b has k elements.

Number of parameters at the encoder side = mV + mk + k^2 + k

At Decoders we have : $E \in R(m \times V),\ U \in R(m \times k), W \in R(k \times k), b \in R(k \times 1) A \in R(V \times k), c \in R(EV \times 1)$

To calculate the total number of parameters at the decoder side, we need to consider the number of elements in each variable and sum them up. The matrix E has m x V elements, the matrix U has m x k elements, the matrix W has k x k elements, the vector b has k elements, the matrix A has V x k elements, and the vector c has V elements.

Number of parameters at the decoder side = mV + mk + k^2 + k + Vk + V

Question 2 :

You will now train your model using any one language from the Aksharantar dataset (I would suggest picking a language that you can read so that it is easy to analyze the errors). Use the standard train, dev, test set from the folder aksharantar_sampled/hin (replace hin with the language of your choice)

BTW, you should read up on how NLG models operate in inference mode, and how it is different from training.

Using the sweep feature in wandb find the best hyperparameter configuration.

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 swept 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:

The sweep was conducted using the following hyperparameters and corresponding values:

Configuration={

epochs: [5,8,10]

hidden_size: [64,128,256,512]

learning_rate: [1e-2,1e-3,1e-1]

bidirectional: [True,False]

cell_type: ['LSTM','GRU','RNN']

encoder_layers: [1,2,3]

decoder_layers: [1,2,3]

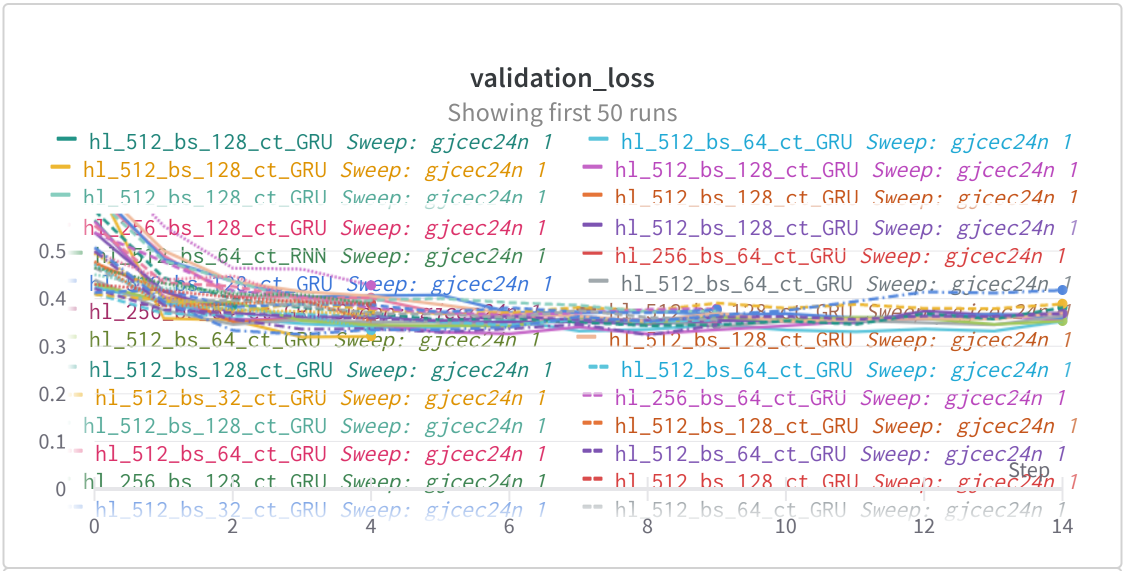
drop_out : [0.0,0.2,0.3]

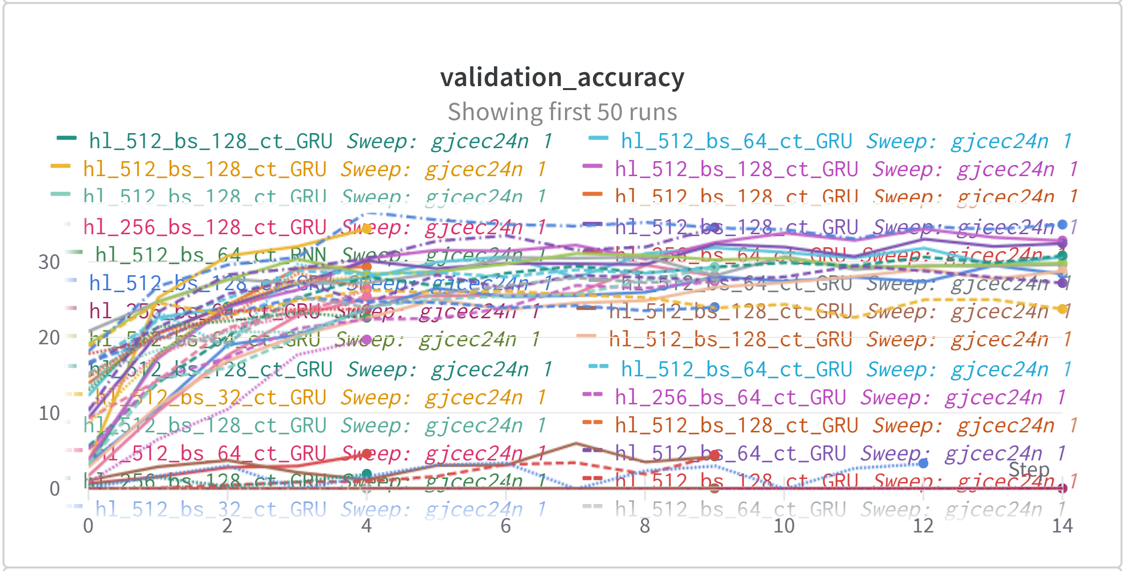
embedding_size: [64,128,256,512]

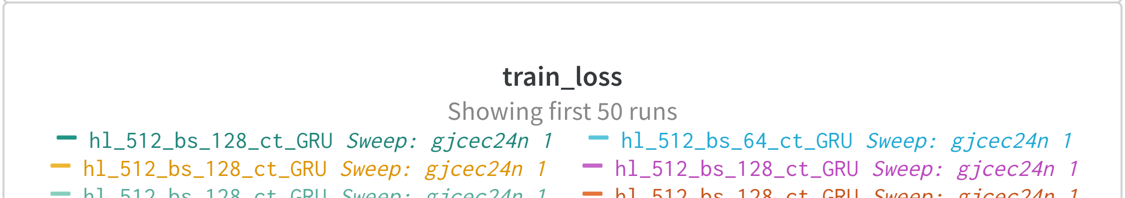
batch_size [32,64,128]

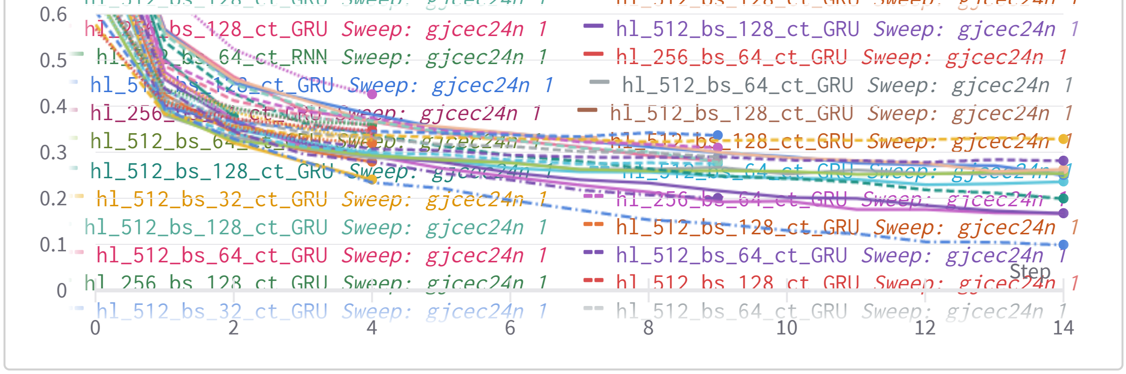
}

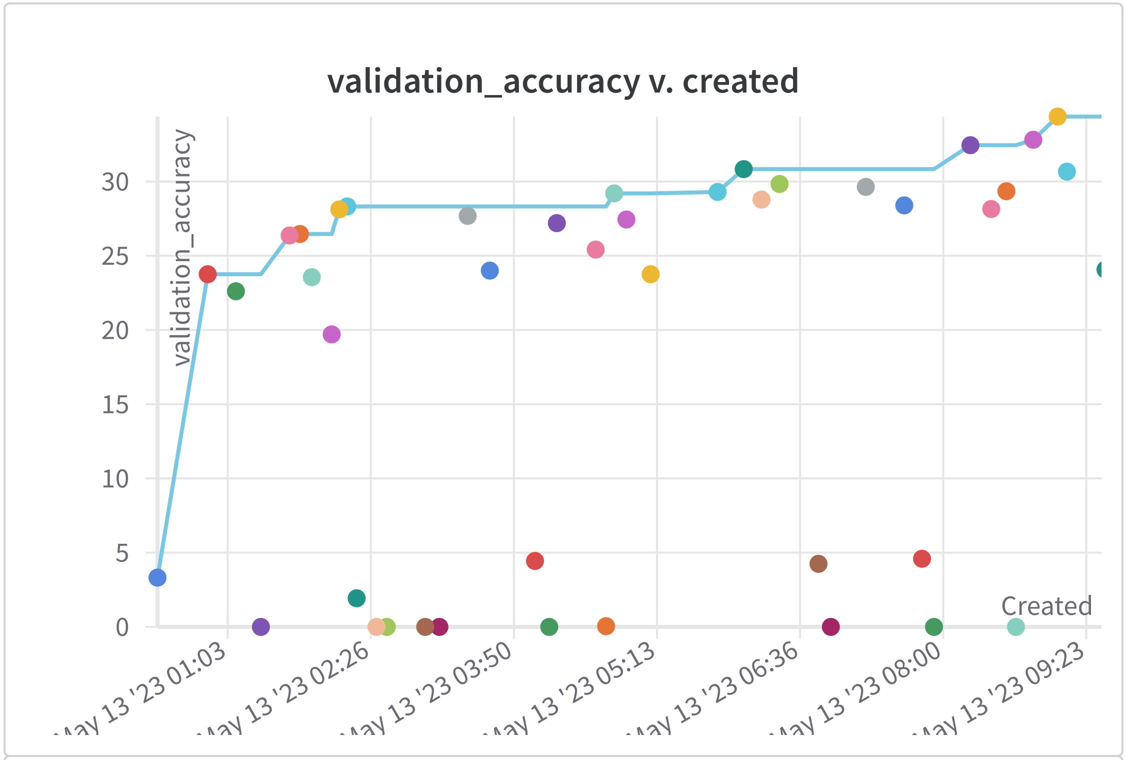
Strategy to minimize the number of Runs: I have used a strategy called Bayesian hyperparameter search instead of relying on grid and random search method.Bayesian hyperparameter search leverages Bayesian inference and probabilistic modeling techniques to intelligently explore the parameter space. Unlike grid and random search, which rely on exhaustive or random sampling respectively, Bayesian optimization utilizes prior knowledge and a probabilistic model to guide the search process.

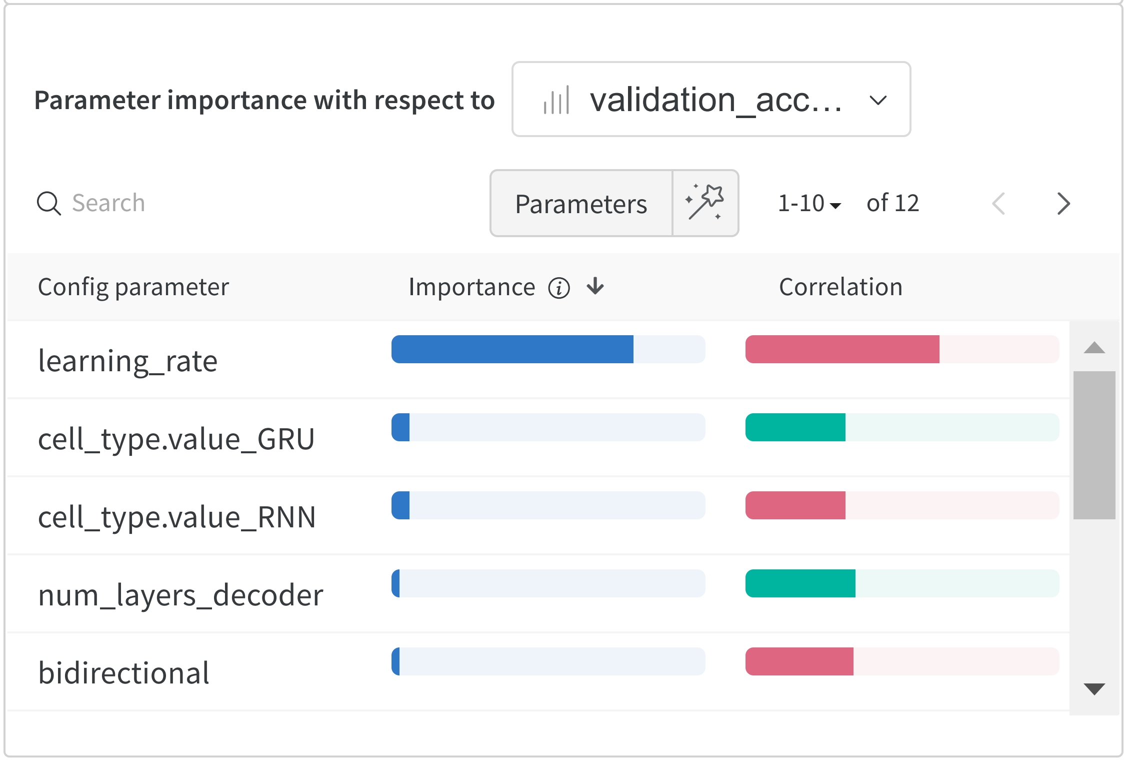








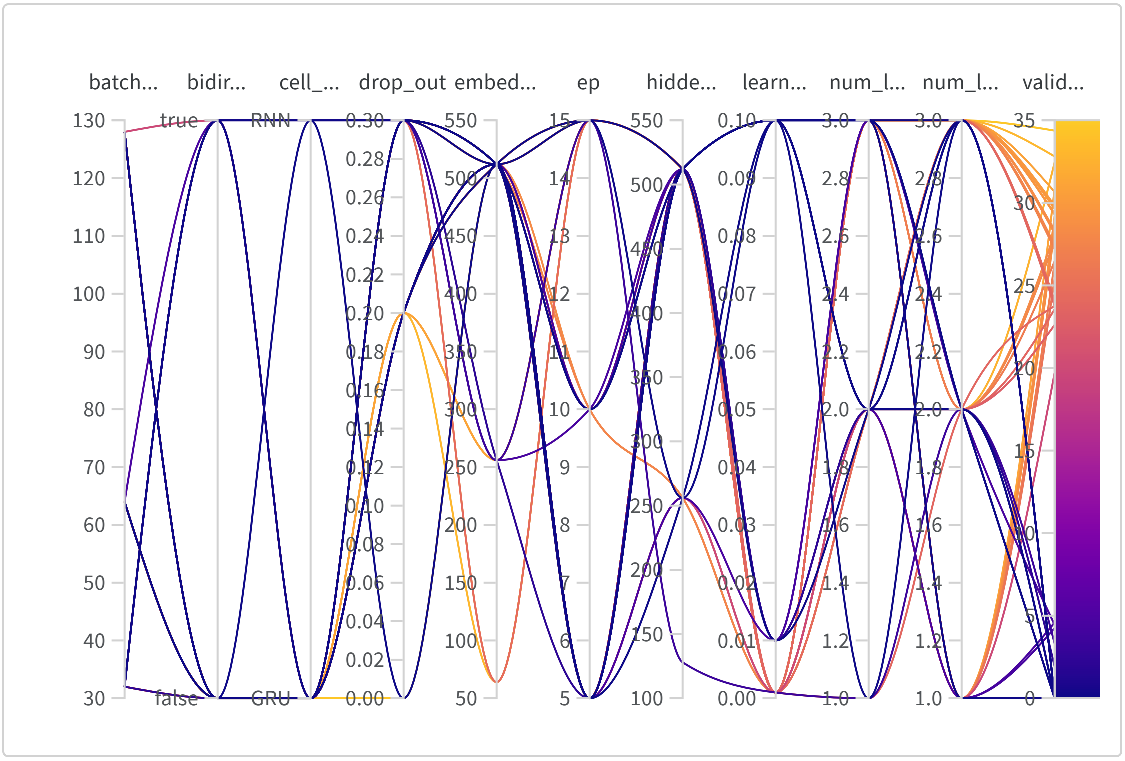



Parameter importance with respect to validation_acc…

Search | Parameters | 1-10 ▾ of 12

Config parameter	Importance ⓘ ↓	Correlation
learning_rate		
cell_type.value_GRU		
cell_type.value_RNN		
num_layers_decoder		
bidirectional		

Note: Run set 2 comprises runs in which the cell type utilized is "LSTM".



Question 3 (15 Marks)

Based on the above plots write down some insightful observations.

Answer:

1.RNN is the worst compared to LSTM and GRU-

All of the RNN models in the sweep had 0% validation accuracy, which means that they were not able to learn the patterns in the data and make accurate predictions. This is likely due to the fact that RNNs are not as good at handling long-term dependencies as LSTMs and GRUs.

2.GRU and LSTM models are more computationally efficient than RNNs.-

RNNs are also more computationally expensive to train and infer than LSTMs and GRUs. This is because RNNs have to keep track of all of the previous states in the sequence, which can be a large number for long sequences.

3.Increasing the number of layers in the decoder increases validation accuracy-

Adding more layers to the decoder can improve the validation accuracy of RNN, LSTM, and GRU models. This is because adding more layers allows the model to learn more complex patterns in the data.

4.GRU outperformed RNN and LSTM-

GRU models performed better than RNN and LSTM models on the validation set. This is likely due to the fact that GRUs are better at handling long-term dependencies than RNNs.

5.The cell type is the most important hyperparameter-

The cell type is the most important hyperparameter for RNN, LSTM, and GRU models. This is because the cell type determines how the model handles long-term dependencies.

6.Dropout leads to better performance as it reduces overfitting-

Dropout is a regularization technique that can be used to reduce overfitting. Overfitting occurs when a model learns the training data too well and is not able to generalize to new data. Dropout works by randomly dropping out (or setting to zero) some of the neurons in the model during training. This forces the model to learn to rely on other neurons, which can help to prevent overfitting.

Overall, the results of the sweep suggest that GRU models are the best choice for sequence learning tasks. GRU models are better at handling long-term dependencies than RNNs and are more computationally efficient. Additionally, dropout can be used to improve the performance of GRU models by reducing overfitting.

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 following hyperparameter produced the highest accuracy on the test set:

configuration={

batch_size: [128],

bidirectional: [False],

cell_type: ['GRU'],

drop_out: [0],

embedding_size: [128],

epochs: [25],

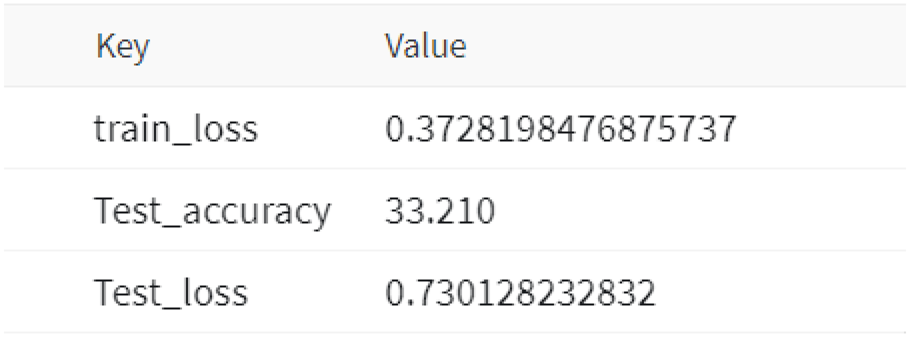
hidden_size: [512],

learning_rate: [0.001],

num_layers_decoder:[3],

num_layers_encoder: [3]

}

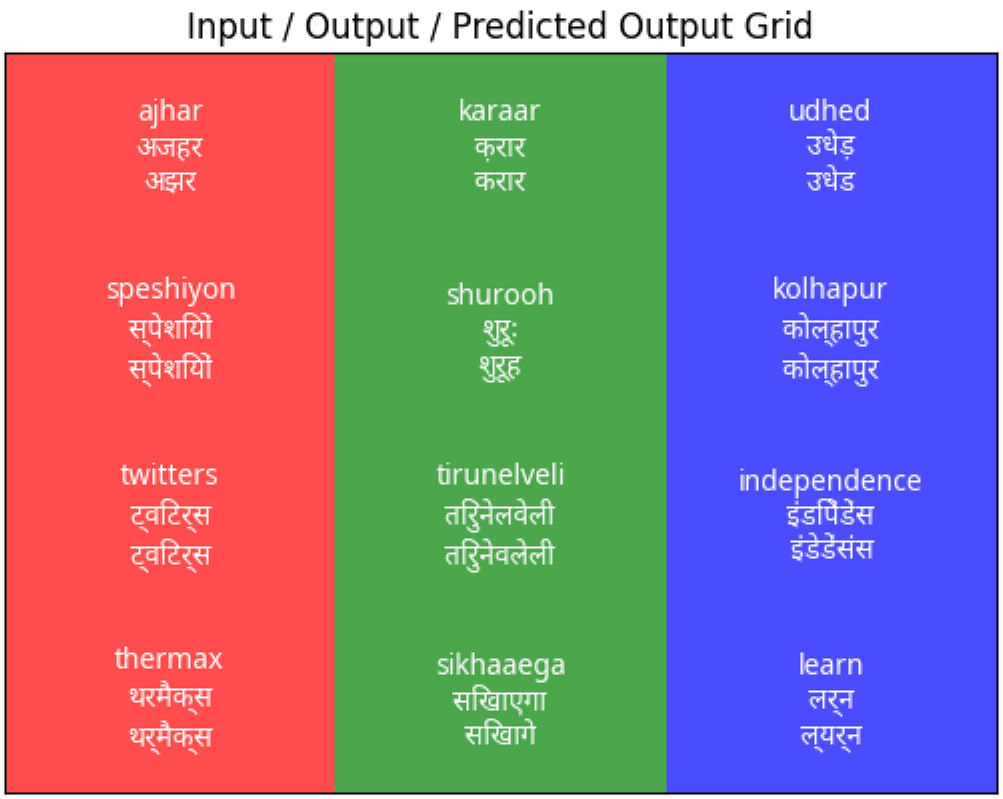


Key	Value
train_loss	0.3728198476875737
Test_accuracy	33.210
Test_loss	0.730128232832

The results obtained from testing this hyperparameter

(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 some test input and prediction samples from the "prediction_vanilla.csv" file, along with the corresponding predictions generated by the best model.



Input / Output / Predicted Output Grid

ajhar
अजहर
अझर

karaar
क़रार
करार

udhed
उधेड़
उधेड

speshiyon
स्पेशियों
स्पेशियों

shurooh
शुरू:
शुरूह

kolhapur
कोल्हापुर
कोल्हापुर

twitters
ट्विटर्स
ट्विटर्स

tirunelveli
तिरुनेलवेली
तिरुनेवलेली

independence
इंडिपेंडेंस
इंडेडेंसंस

thermax
थरमैक्स
थर्मैक्स

sikhaaega
सिखाएगा
सिखागे

learn
लर्न
ल्यर्न

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

Answer.

- The model often encounters challenges when dealing with English words that have multiple pronunciations, such as the word "learn." In English, it can be pronounced as "lɜːrn" or "lɑːrn," depending on the accent or dialect. When the model tries to transliterate or represent these pronunciations in Hindi, it face difficulties in accurately capturing the specific sounds and variations. This showcases the complexity of translating and representing different pronunciations across languages, especially when there are subtle differences in phonetics and accents
- The model encounters difficulties in predicting the correct handling of partial characters in words like "brigt" and "fornat." It struggles to determine whether to join or separate the letters in such cases.

Question-5 : (20 Marks)

Add an attention network to your base 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 the appropriate plots below.

Answer. Yes,we tuned the hyperparameters again.

The sweep was executed using the following hyperparameters and corresponding valuue:

Configuration:{

'hiddenSize' : [128,256,512]

'num_layers':[1]

 'learning_rate': [1e-2,1e-3]

 'cell_type': ['LSTM','RNN','GRU']

 'drop_out': [0.0]

 'embedding_size': [64,128,256,512]

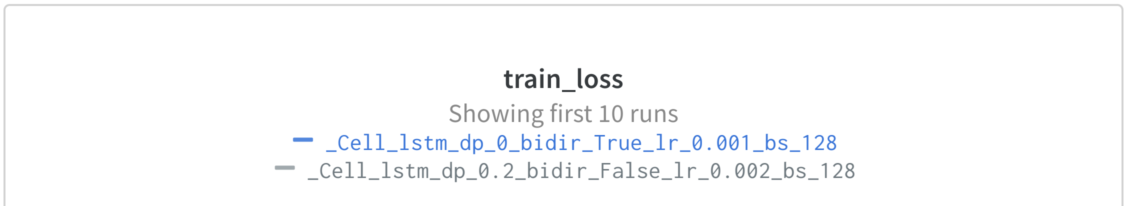
 'batch_size':[32,64,128]

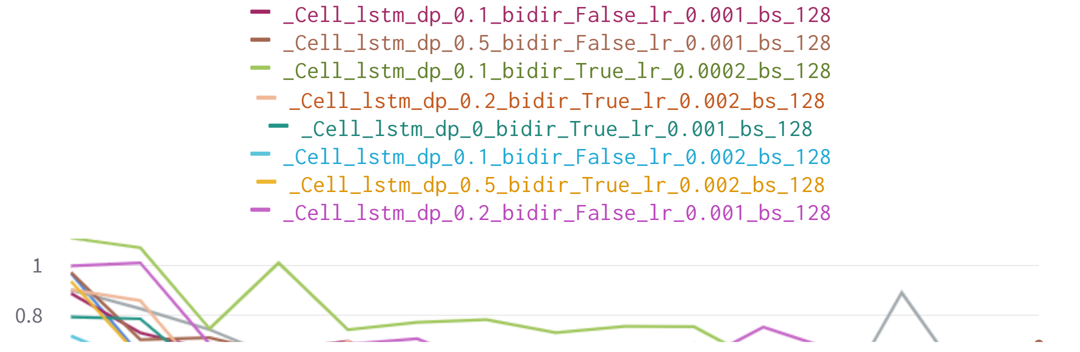
'bi_directional': [True,False

'epochs':[5,10,15,20]

  }

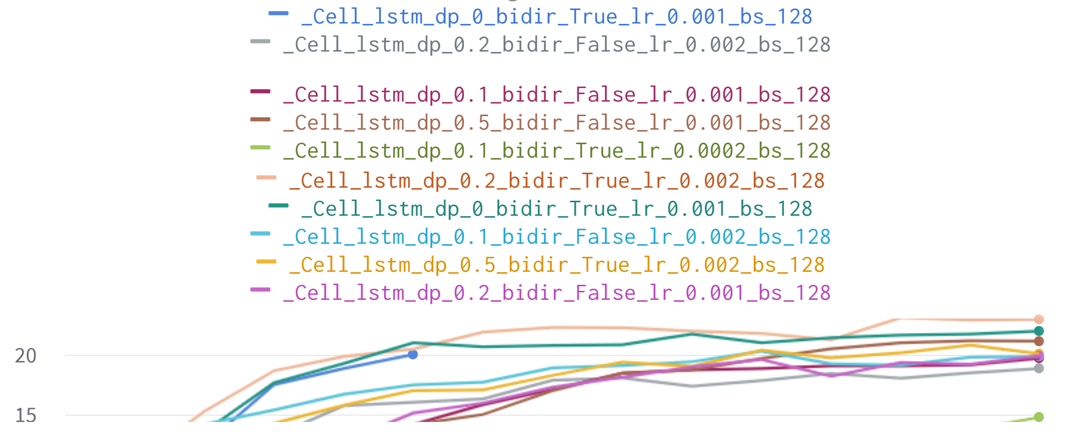
The reported accuracies pertain to the scenario in which the attention model is utilized with a one-layer configuration for both the decoder and the encoder.





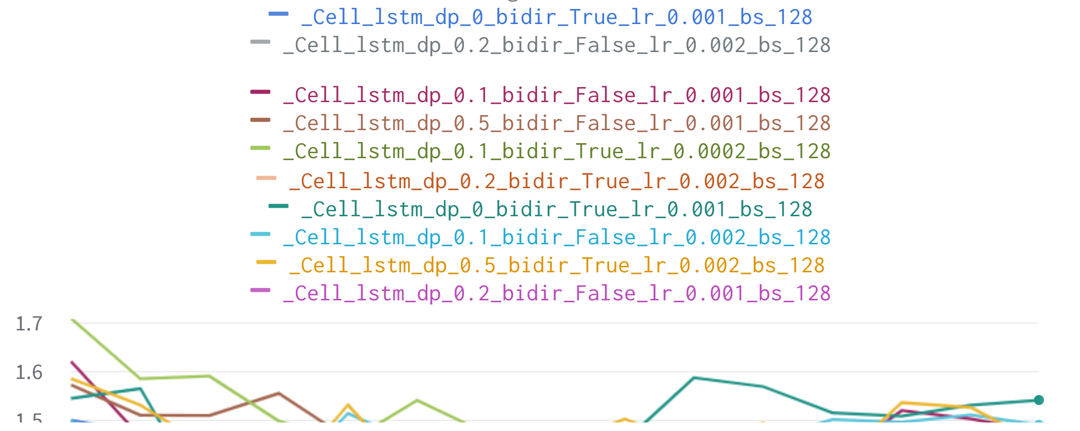
val_acc

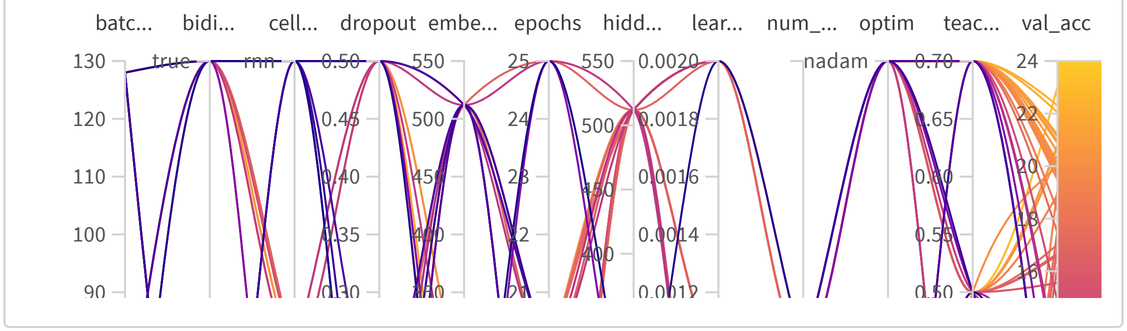
Showing first 10 runs



val_loss

Showing first 10 runs





(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 Hyeperparameters for the test set is :

Configuration:{

batchSize : 128,

biDirectional : True

cellType: 'LSTM'

dropOut: 0

embeddingSize: 64

epochs : 15

hiddenSize : 512

learningRate : 0.002

encoderLayers :1

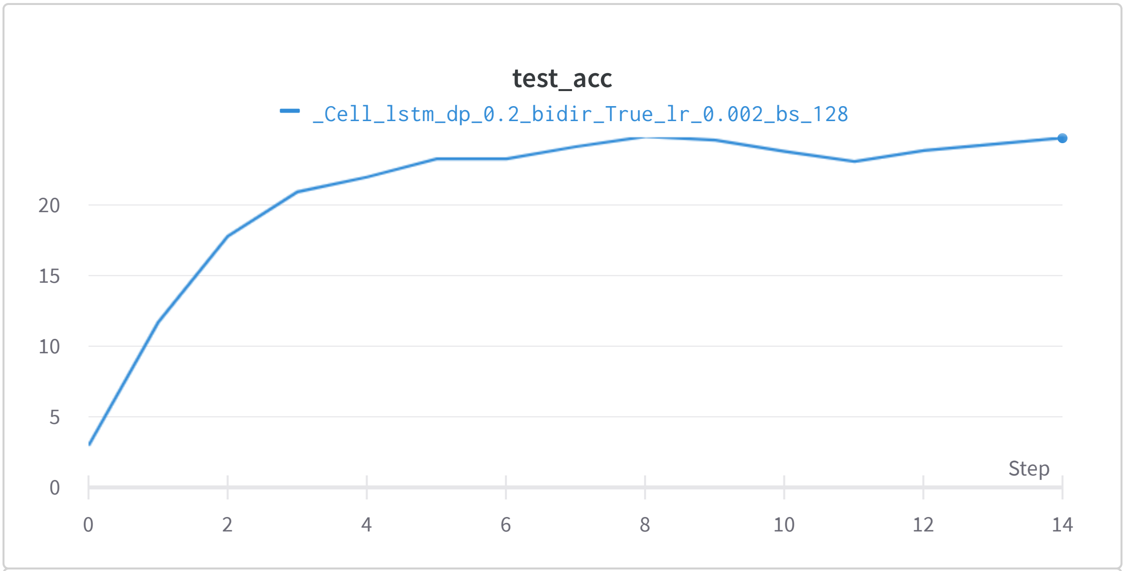
decoderLayers :1

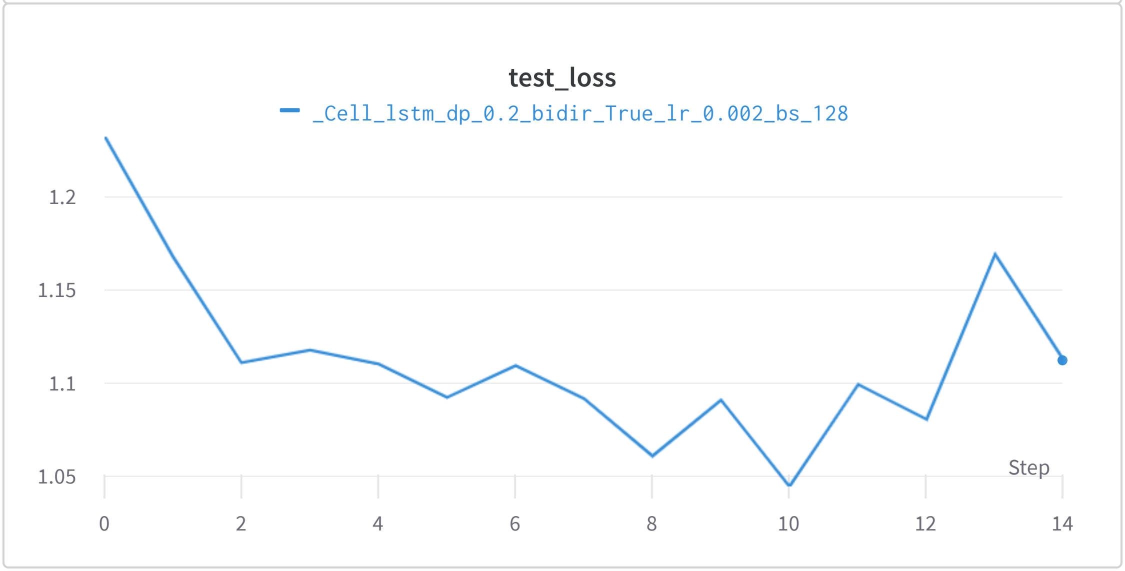
optimizer : Adam

teacherForcingRatio: 0.5

}

When the model is executed with the addition of an attention model and both the encoder and decoder are comprised of a single layer, the test accuracy is: 24.7314453125 and the test loss is: 1.112295538187027.





(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

Answer: Indeed, when we take into account the utilization of a single layer encoder and decoder in the case of the attention model with that of a single layer encoder and decoder in the vanilla model , it becomes apparent that the performance of the attention-based model surpasses that of the vanilla model. This notable improvement can be primarily ascribed to two key factors:

- Firstly, the attention-based model addresses the issue of remembrance to a considerable degree by directing its focus to the pertinent input words at each timestep, rather than processing the input as a whole.
- Secondly, the model effectively incorporates the influence of neighboring characters when predicting the output at a given timestep. Consequently, it excels in scenarios where the same English alphabet input possesses different pronunciations.

Question 7 :(10 Marks)

Paste a link to your GitHub Link:https://github.com/AvinashKushwah/cs6910_assignment-3

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 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.

Self-Declaration

I, **Avinash Singh Kushwah (Roll no: CS22M024)**, 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.