CS6910 - Assignment 3

Use recurrent neural networks to build a transliteration system.

Viswanathan S

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 m, the 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)

Here's the breakdown:

1. Encoder RNN:

- Input embedding: m computations

- GRU cell: 3k computations (reset gate, update gate, new memory content)

- T time steps

Total computations for the encoder: T * (m + 3k)

2. Decoder RNN:

- Input embedding: m computations

- GRU cell: 3k computations (reset gate, update gate, new memory content)

- Output linear layer: V * k computations

- Softmax: V computations

- T time steps

Total computations for the decoder: T * (m + 3k + V * k + V)

Overall, the total number of computations for the network without the attention mechanism is:

Total computations = T * (m + 3k) + T * (m + 3k + V * k + V)

This computation estimate assumes a single layer for both the encoder and decoder. If the network architecture includes multiple layers, the computation would increase accordingly.

(b) What is the total number of parameters in your network? (assume that the input embedding size is m, the 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)

To determine the total number of parameters in the network, we need to consider the parameters in each component of the network:

1. Encoder RNN:

- Input embedding: m * k parameters

- GRU cell: (m + k) * 3k parameters

Total parameters for the encoder: (m * k) + ((m + k) * 3k)

2. Decoder RNN:

- Input embedding: V * k parameters

- GRU cell: (V + k) * 3k parameters

- Output linear layer: V * k parameters

Total parameters for the decoder: (V * k) + ((V + k) * 3k) + (V * k)

Overall, the total number of parameters in the network is the sum of the parameters in the encoder and decoder:

Total parameters = (m * k) + ((m + k) * 3k) + (V * k) + ((V + k) * 3k) + (V * k)

This parameter estimate assumes a single layer for both the encoder and decoder. If the network architecture includes multiple layers, the number of parameters would increase accordingly.

Question 2 (10 Marks)

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. This blog might help you with it.

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
- bidirectional: Yes, No
- dropout: 0.2, 0.3 (btw, where will you add dropout? You should read up a bit on this)
- beam search in decoder with different beam sizes:

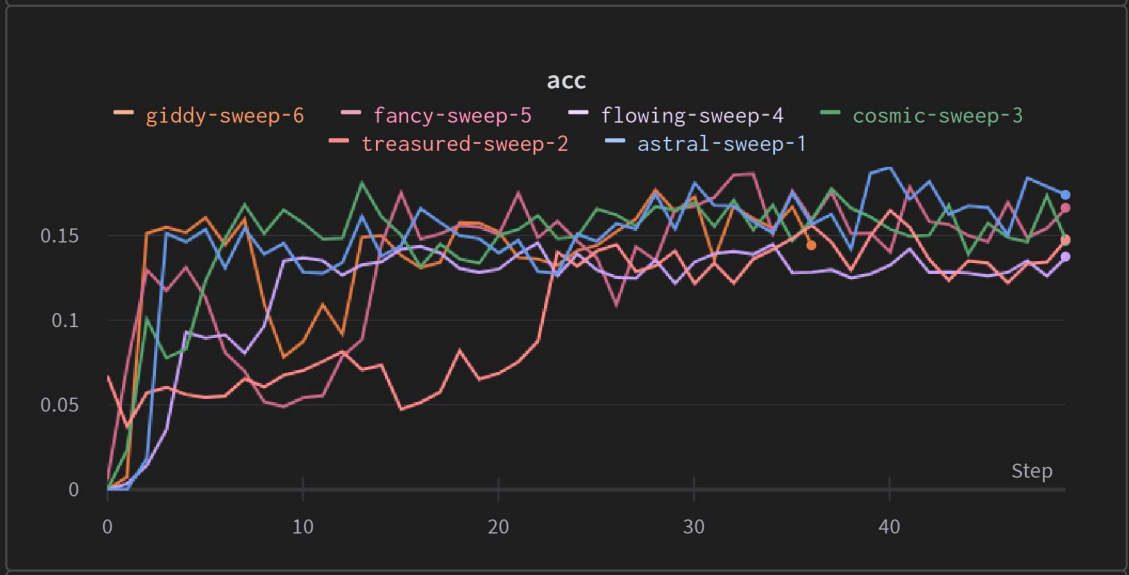
Based on your sweep, please paste the following plots, which are automatically generated by wandb:

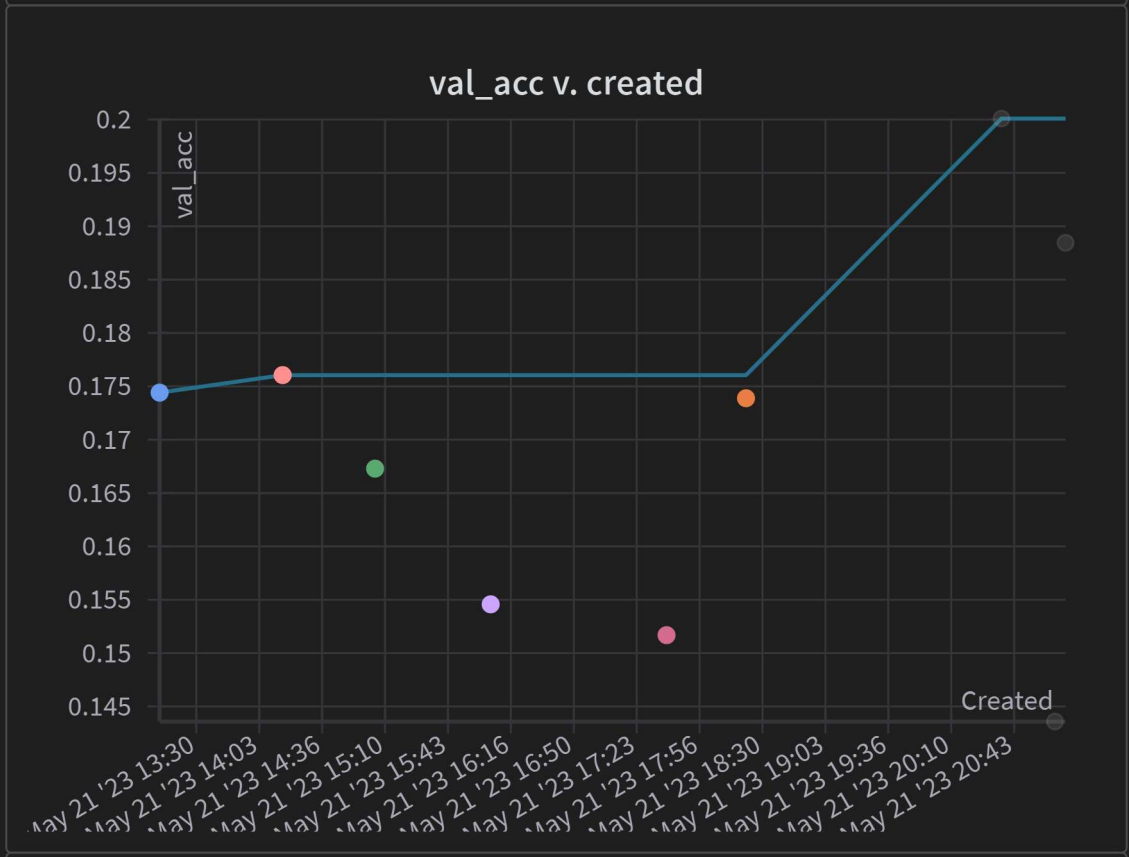
- accuracy v/s created plot (I would like to see the number of experiments you ran to get the best configuration).
- parallel co-ordinates plot

- correlation summary table (to see the correlation of each hyperparameter with the loss/accuracy)

Also, write down the hyperparameters and their values that you 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.

I used the mentioned set of hyperparameters to generate sweep config.





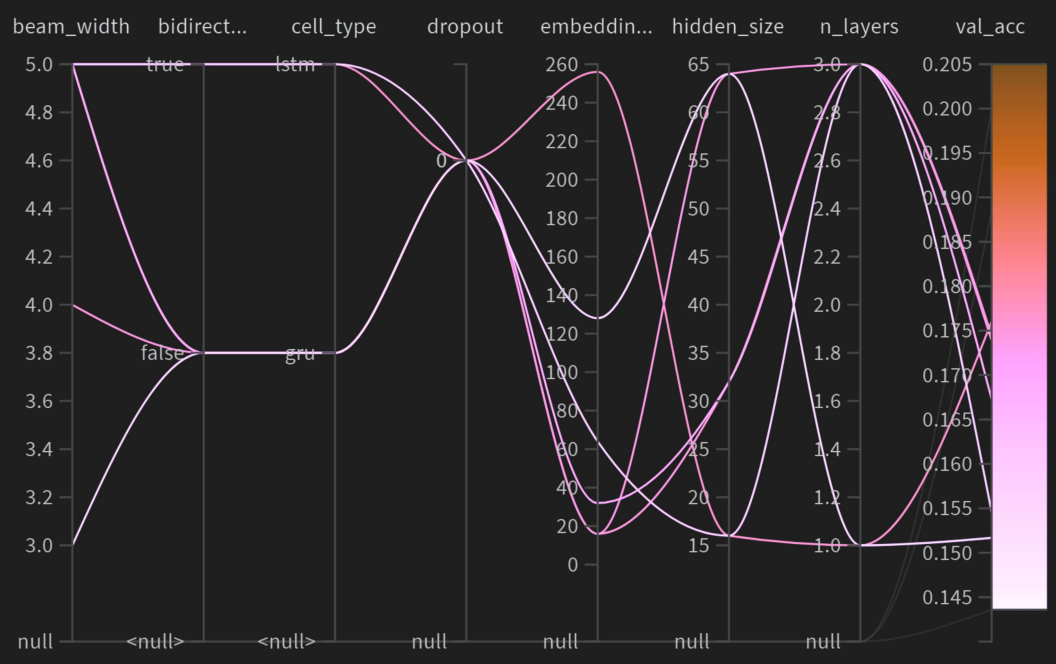
Parameter importance with respect to val_acc

Search

Parameters

1-0 of 0

Loading...


beam_width
bidirect...
cell_type
dropout
embeddin...
hidden_size
n_layers
val_acc
5.0
4.8
4.6
4.4
4.2
4.0
3.8
3.6
3.4
3.2
3.0
true
false
lstm
gru
0
260
240
220
200
180
160
140
120
100
80
60
40
20
0
65
60
55
50
45
40
35
30
25
20
15
3.0
2.8
2.6
2.4
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.205
0.200
0.195
0.190
0.185
0.180
0.175
0.170
0.165
0.160
0.155
0.150
0.145
null
<null>
<null>
null
null
null
null

Run set 6

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 are true. I just wrote some random comments that came to my mind)

Here are some insightful observations based on different hyperparameters:

1. Input Embedding Size: Increasing the input embedding size for transliteration can help the model capture more fine-grained phonetic and orthographic features of the input words. A larger embedding size allows the model to represent a wider range of characters and their combinations, enabling better learning of transliteration patterns. However, it's important to avoid excessively large embedding sizes that may lead to overfitting or increased computational complexity.

2. Number of Encoder-Decoder Layers: Increasing the number of encoder and decoder layers in the transliteration model can enhance its ability to capture complex relationships between source and target languages. Additional layers provide the model with more hierarchical representations and allow it to learn intricate mapping patterns. However, increasing the number of layers also increases the risk of overfitting, especially when training data is limited. Regularization techniques like dropout can help mitigate overfitting concerns.

3. Hidden Layer Size: The size of the hidden layers in the transliteration model influences its capacity to capture nuanced mapping rules between languages. Larger hidden layer sizes enable the model to learn more complex relationships and capture fine-grained details of transliteration. However, using excessively large hidden layer sizes may lead to overfitting or increased computational requirements. Choosing an appropriate hidden layer size is crucial to balance model complexity and generalization capacity.

4. Cell Type: The choice of cell type in the transliteration model, such as LSTM, GRU, or simple RNN, impacts its ability to capture and model transliteration patterns. LSTM and GRU cells, with their gating mechanisms, are particularly effective in handling sequence-to-sequence tasks like transliteration. They can capture long-range dependencies and mitigate the vanishing/exploding gradient problem, leading to more accurate and reliable transliteration results compared to simple RNN cells.

5. Bidirectional Encoding: Bidirectional encoding can be beneficial in the case of transliteration, as it allows the model to leverage context from both the source and target languages. By processing the input sequence in both forward and backward directions, the model can capture orthographic and phonetic cues from both sides, improving its understanding of the transliteration task. Bidirectional encoding is especially useful when the context from both directions is important for accurate transliteration.

6. Dropout: Incorporating dropout regularization during training of the transliteration model can help prevent overfitting and improve generalization. Dropout introduces noise by randomly disabling neurons during each training iteration, forcing the model to rely on different subsets of features and reducing interdependencies among neurons. This regularization technique can enhance the transliteration model's ability to handle diverse input variations and improve its performance on unseen data.

7. Beam Search in Decoder: Beam search is a decoding strategy commonly used in sequence-to-sequence models, including transliteration. Applying beam search in the decoder allows the model to explore multiple hypotheses during the transliteration process. Choosing different beam sizes can impact the trade-off between output quality and computational complexity. Larger beam sizes increase the exploration space but require more computational resources, while smaller beam sizes may yield faster decoding but potentially miss out on better transliteration candidates.

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).

Got an accuracy of 29.68%

(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.

Provided in repo

(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 maybe it's just me!
- ...

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.

No I did not tune.

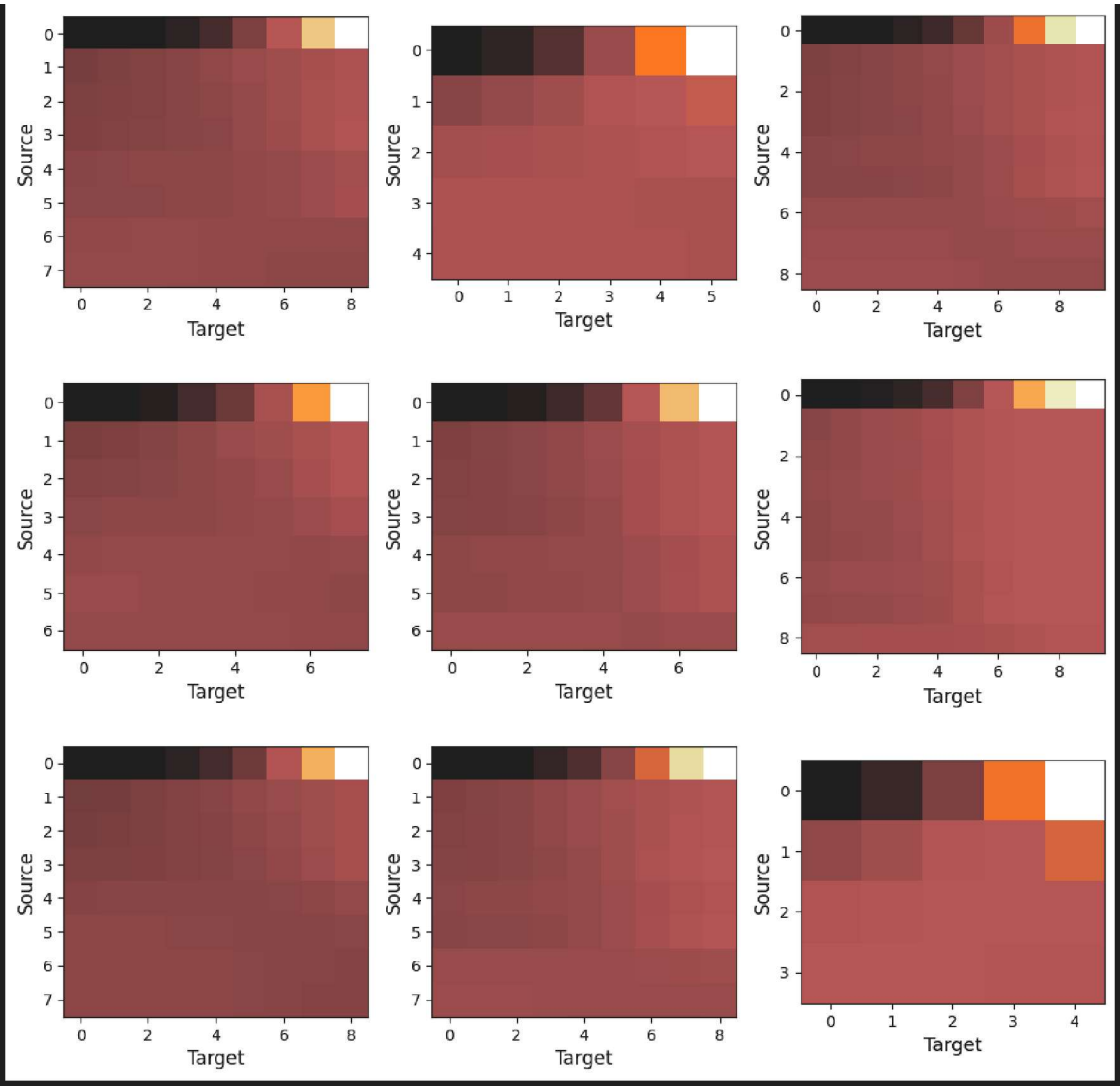
(b) Evaluate your best model on the test set and report the accuracy. Also, upload all the predictions on the test set in a folder `predictions_attention` on your GitHub project.

Provided in the repo.

(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 that were predicted incorrectly by your best seq2seq model are predicted correctly by this model)

No. I was not able to train the attention based model properly. But In general the attention based model should perform better.

(d) In a 3×3 grid, paste the attention heatmaps for 10 inputs from your test data (read up on what attention heatmaps are).



(UNGRADED, OPTIONAL) Question 6 (0 Marks)

This a challenging question, and most of you will find it hard. Note that this question has no marks and will not be graded. This is only for students looking for a challenge and wanting to get something more out of the course.

I like the visualization in the figure captioned "Connectivity" in this article. Make a similar visualization for your model. For some starter code, please look at this blog. 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?

Question 7 (10 Marks)

Paste a link to your GitHub Link :
https://github.com/Vishwanath1999/cs6910_ssignment_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.

(UNGRADED, OPTIONAL)
Question 8 (0 Marks)

Note that this question has no marks and will not be graded. This is only for students looking for a challenge and wanting 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!

Self-Declaration

I, S VISWANATHAN (Roll no: EE21S075), 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/viswa_ee/CS6910_NLG/reports/CS6910-Assignment-3--Vmlldzo0NDI2MzYx