

# ABSTRACT

This project aims to evaluate different approaches to neural machine translation, incorporating deep learning techniques such as sequence to sequence models, LSTM cells and attention models. As part of this project we made an in-depth analysis of each approach and an error analysis of translations made by the model. The accuracy metrics used are BLEU scores. The models build perform English to Hindi translation.

# INTRODUCTION

## The Problem

The inability to communicate due to language barriers is one of the largest impediments we dace as a society today. Language Translation is a task with the potential for global impact. We can even see the importance in business. In order to overcome the language barrier in the globalized society, we chose to create a language translation model.

## Encoder-Decoder Models

Encoder-decoder models are the simplest version of Neural Machine Translation(NMT).

The idea is relatively simple: we read in the words of a target sentence one-by-one using a recurrent neural network, then predict the words in the target sentence.

# DATA & APPROACH

## DATA SET

This dataset is primarily intended for translation tasks from English to Indian Language Hindi.

The data set we used as part of this project is downloaded from

<http://lotus.kuee.kyoto-u.ac.jp/WAT/indic-multilingual/index.html>

The data set has 84557 lines which consists of 17906 unique English words and 21973 unique Hindi words.

The examples come in source-target sentence pairs, in the following format

Source sentence : You was lucky there, weren't you?

Target Translation : तुम्‍हारी किस्‍मत अच्‍छी थी, नहीं थी क्‍या?

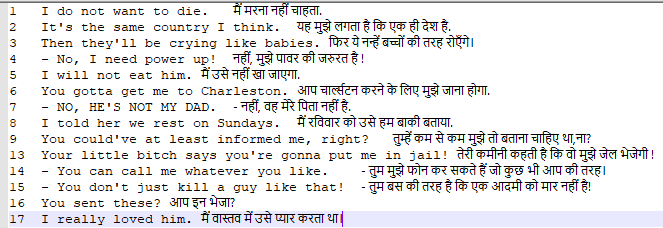
## APPROACH

## DATA PREPROCESSING:

Data preprocessing is the initial step in this project.

English and Hindi Text preprocessing has been done removing the nonascii characters from the text. The Cleaned data set ~ 74k lines are used in this project.

~10000 English text lines omitted from target file and corresponding Input file for the Sequence to Sequence models.

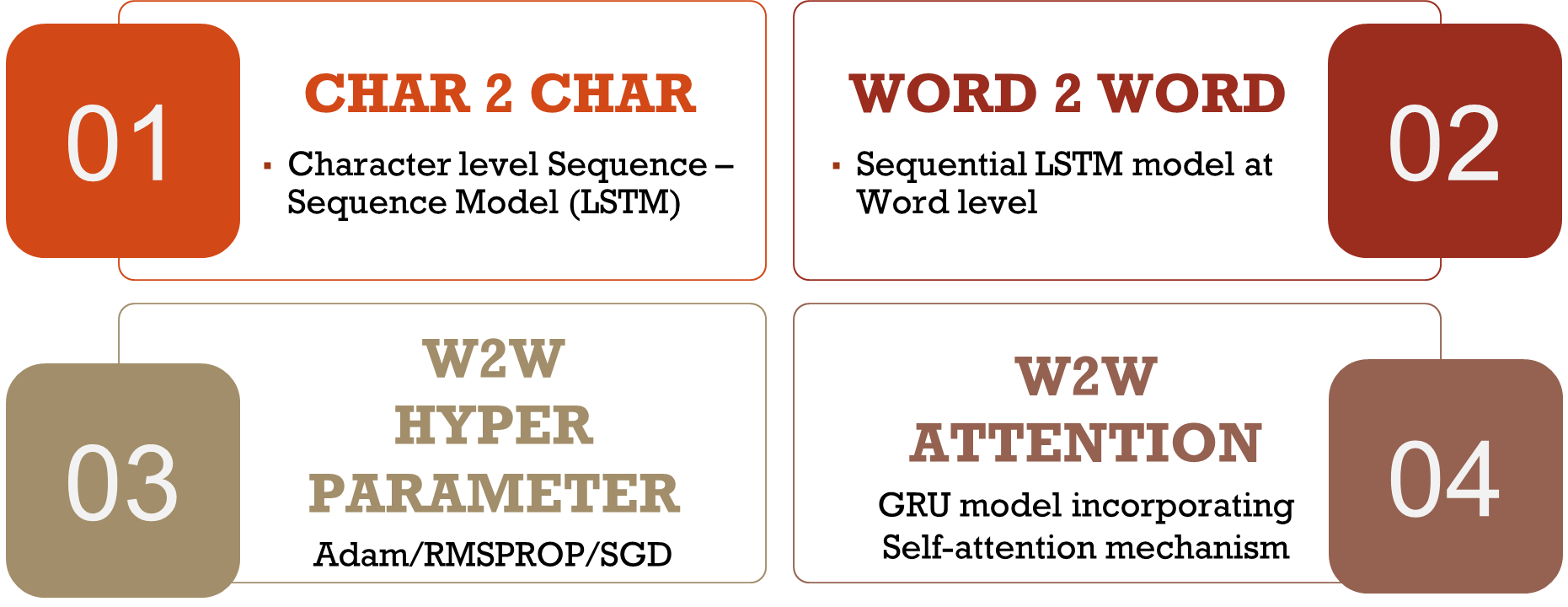


2.2.2 MODEL DEPLOYMENT AND EVALUATION

Build the model using various models passing the relevant data as inputs.

Train the model and evaluate using BLEU scores.

# MODELS



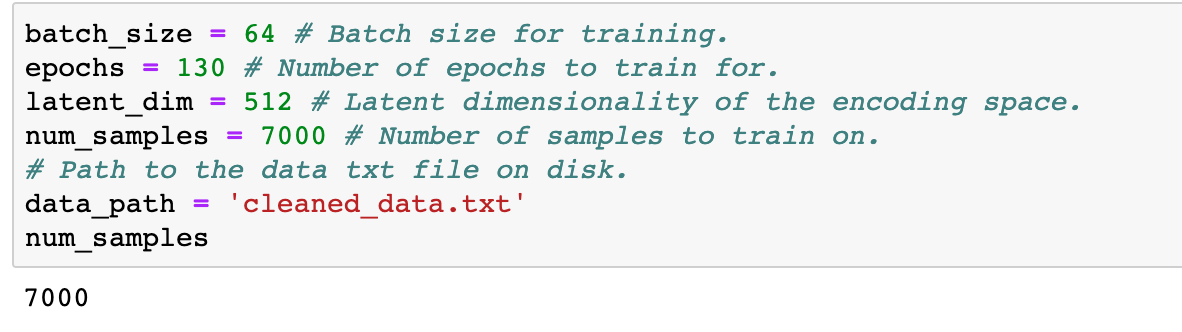
## CHAR 2 CHAR

We applied a simple character seq 2 seq model for the text corpus for English to Hindi machine translation.

In English we have representation of each letters in the alphabet, but in Hindi there is no exact translation for the letters. We will describe the coding approach for the character to character model as the other models are just a variation where tokenized words are passed instead of letter-wise inputs.

Firstly, we will train the model and then look at the inference models on how to translate English sequence to Hindi sequence (used for predicting on the input sequence).

Since the sentences are large, we will take some samples out of the whole to train, such as 7000.



We vectorize the data because vectorization divides the computation times by several order of magnitudes and the difference with loops increase with the size of the data. Hence, if you want to deal with large amount of data, rewriting the algorithm as matrix operations may lead to important performances gains.

A screenshot of a cell phone

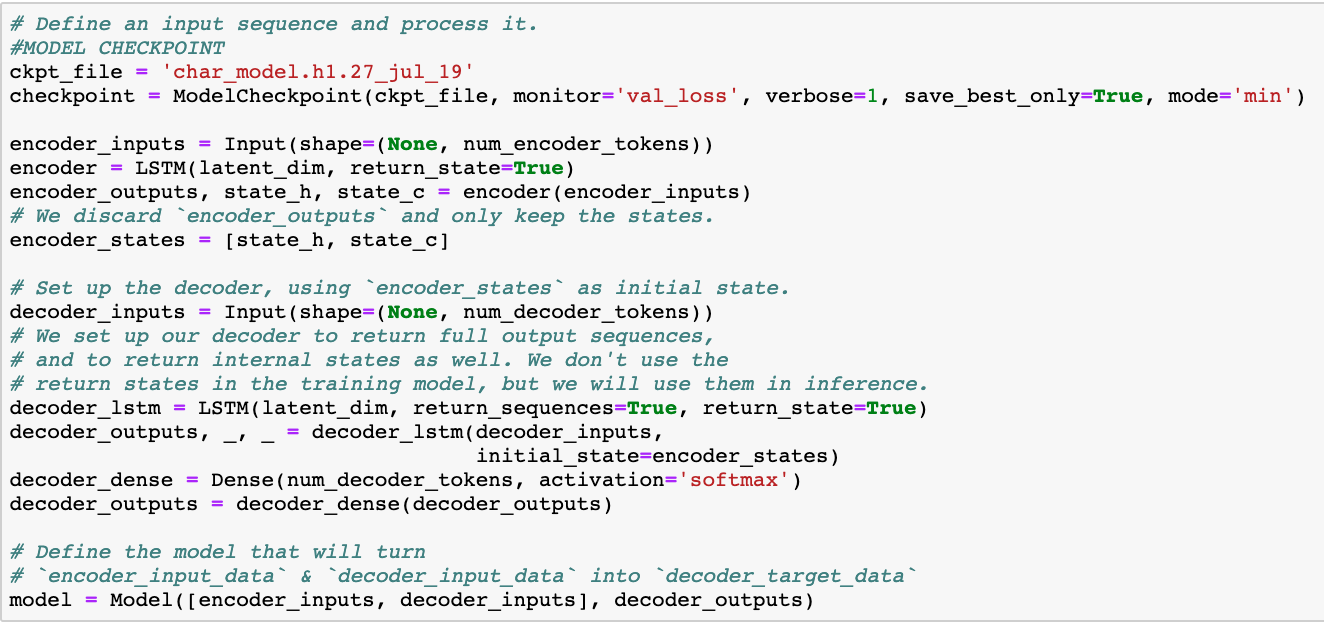
Description automatically generated

We define the input sequence and process it by Encoder Decoder architecture.

Both encoder and the decoder are LSTM models

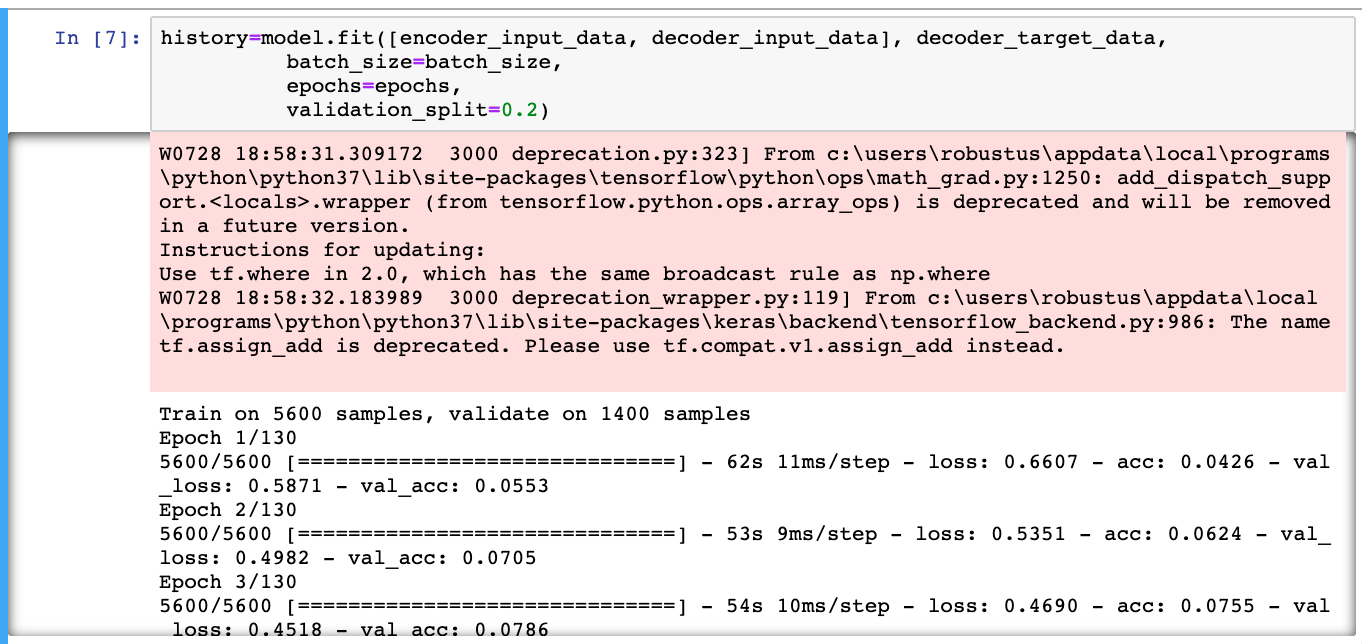
Encoder reads the input sequence and summarizes the information in something called as the internal state vectors (in case of LSTM these are called as the hidden state and cell state vectors). We discard the outputs of the encoder and only preserve the internal states.

Decoder is an LSTM whose initial states are initialized to the final states of the Encoder LSTM. Using these initial states, decoder starts generating the output sequence.



A screenshot of a social media post

Description automatically generated



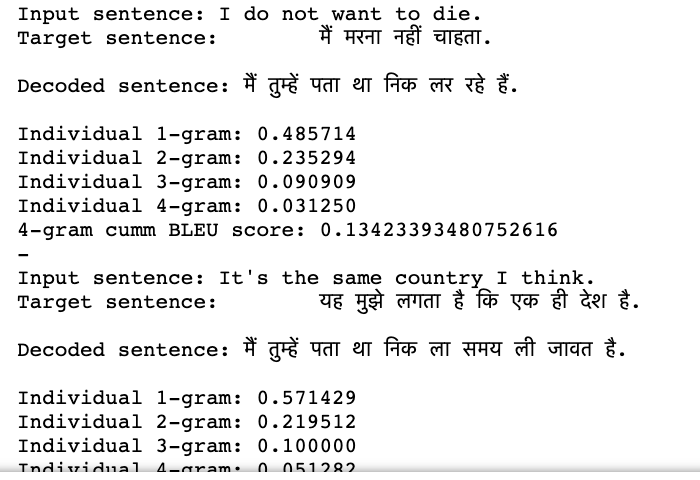


Here we receive the plots for the loss and accuracy, and we can observe that width within the training and validation samples.

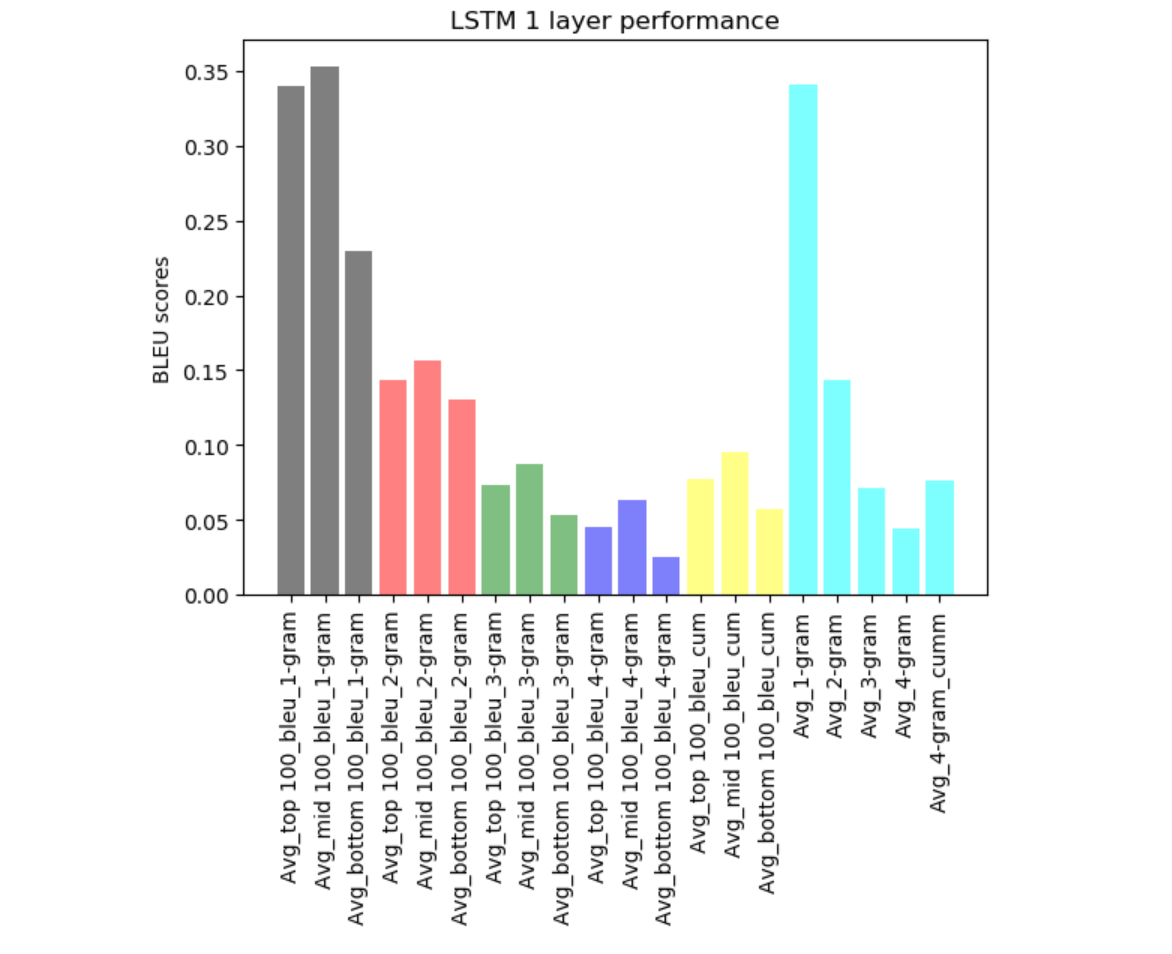
We now decode with the inference method for the samples from the dataset.

A screenshot of a social media post

Description automatically generated



Here we receive the sentences from the model and obtain the Bleu scores such that 0 and 1 represents the scale of correct translation. The max observed it 0.35 Bleu score.



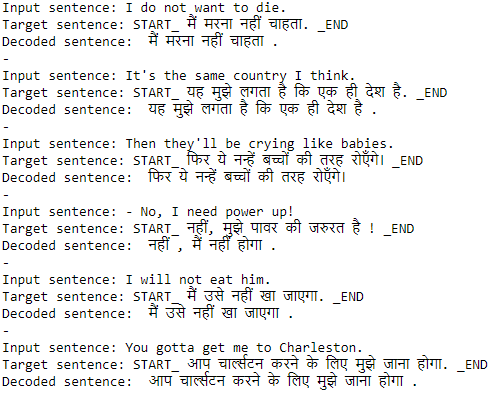
## WORD 2 WORD

After using the character to character model with limited success over 120 epochs we decided to model a variant of traditional LSTM encoder decoder model by stacking another layer of LSTM encoder and additional decoder. The difference in approach is also that the inputs passed are tokenized word versions of sentences instead of them being passed letter by letter. Hence, the decoder output will be artificially padded with white spaces to make it legible in sentence form.

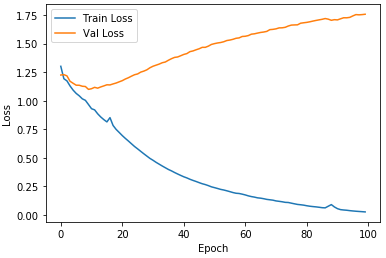
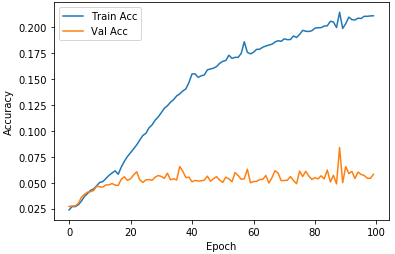
Over various iterations, the best performing model was recorded as follows:

* + - 512 latent dimensions
    - Batch size 64
    - ADAM optimizer
    - 100 epochs
    - 7000 samples

It incorporated the best model checkpoint call back basis validation loss.

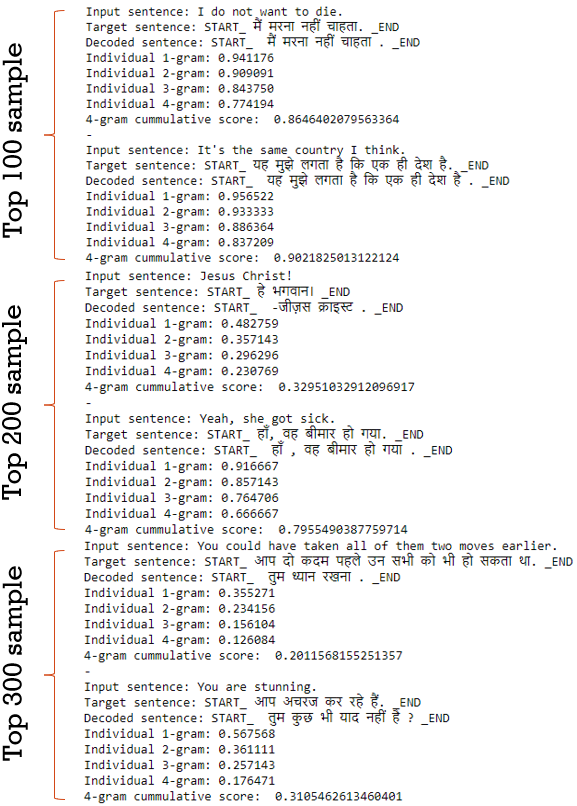


The model was run over various epochs to understand where it would give practically high results without overfitting the model. The accuracy and loss performances are as follows:



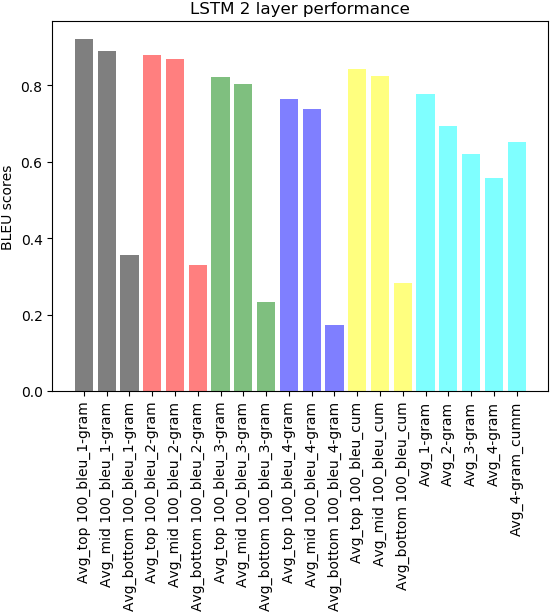
We can see that while the training accuracy increases, the validation accuracy lingers around 0.05 and while training loss decreases, the validation loss shows a gradual increase.

However, since accuracy and loss are not sufficient metrics to determine the performance of the models, we resorted to BLEU score. The BLEU score was calculated for individual 1-gram, 2-gram, 3-gram, 4-gram and cumulative 4-gram model.



On first glance, the machine translation looks excellent however we decided to gauge the performance over 300 samples – top 100 samples, mid 100 samples and tailing 100 samples to see variations in learning the words across the training sample set. This was then calculated for each of the above metrics as well as averaged over all samples to see overall performance.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **N-gram** | **Top 100** | **Mid 100** | **Bottom 100** | **All 300** |
| 1 | 0.92 | 0.89 | 0.36 | 0.78 |
| 2 | 0.88 | 0.87 | 0.33 | 0.69 |
| 3 | 0.82 | 0.80 | 0.23 | 0.62 |
| 4 | 0.77 | 0.74 | 0.17 | 0.56 |
| 4- gram cumulative | 0.84 | 0.83 | 0.28 | 0.65 |



To summarize, the model works better with smaller sentences and learns faster for the top samples. This performance goes down as it moves down the sample. Perhaps more epochs could offset this change in performance. Hence the BLEU scores are excellent for the top 100 samples, average for mid-100 samples and dismal for the tailing 100. However, the average BLEU cumulative score of 0.65 sets a good benchmark for the performance of the word to word LSTM dual layer configuration.

## WORD 2 WORD HYPER PARAMETER TUNING

The tune-ability of an algorithm, hyperparameter, or interacting hyperparameters is a measure of how much performance can be gained by tuning it. For an LSTM, the learning rate followed by the network size are its most crucial hyperparameters.

  Hyperparameters refer to the parameters that are set before the start of training.

* Optimizers
* Learning rate
* Type of attention
* Attention Architecture
* Inference mode

We tried using RMS Prop optimizer and with the below hyperparameters, achieved an average cumulative BLEU score of 0.29 .

**3.3.1 USING RMSProp optimizer**

    Latent Dimension : 512

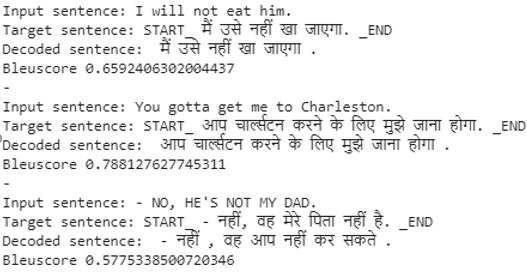
    Epoch : 100

    Sample : 7000

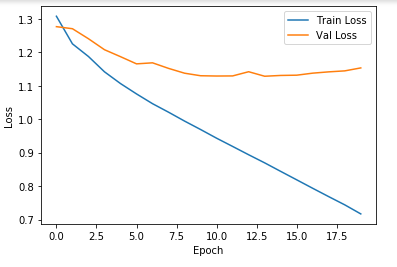
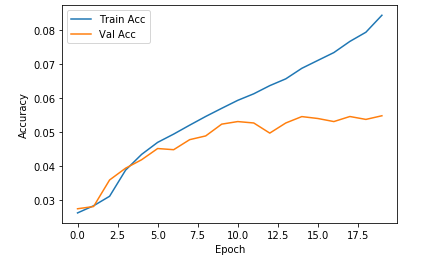
    Batch Size : 64

    Avg Cumulative Blue Score : 0.69

The Translations turned out from this model are as below:



Accuracy and Loss performances are as follows:

**3.3.2 Using SGD Optimizer**

We ran the model using SGD Optimizer adding drop out to enhance the performance. SGD optimizer with latent dimensions 256 and batch size 64 had performed well on this data.

Below are the performance graphs for loss and accuracy values using SGD optimizer.

SGD optimizer  with Drop out (round 1)

    Latent Dimension : 512

    Epoch : 50

    Sample : 7000

    Batch Size : 128

    Max Cumulative Blue Score : 0.05

  SGD optimizer  with Drop out (round 2)

 Latent Dimension : 256

    Epoch : 70

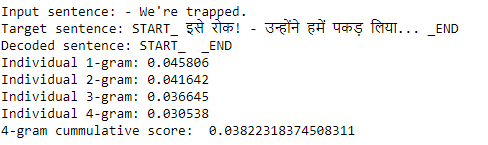
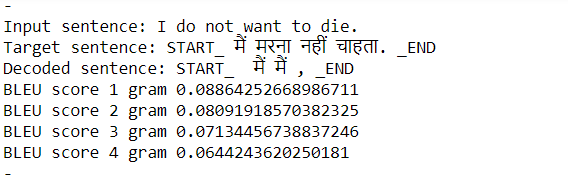
    Sample : 1000

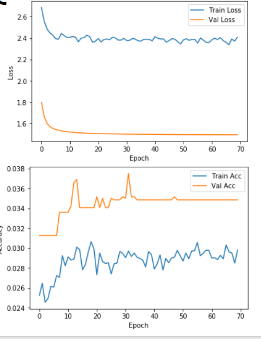
    Batch Size : 64

    Max Cumulative Bleu Score : 0.512

    Avg Cumulative Bleu Score : 0.244

Below are the sample translations from the model.





**3.3.3 Using ADAM optimizer**

Model is built on ADAM optimizer with hyperparameters

Latent Dimension : 512

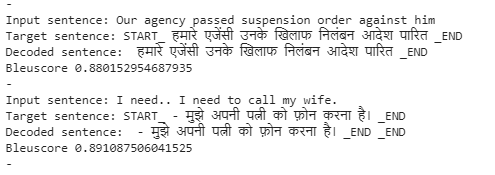
    Epoch : 100

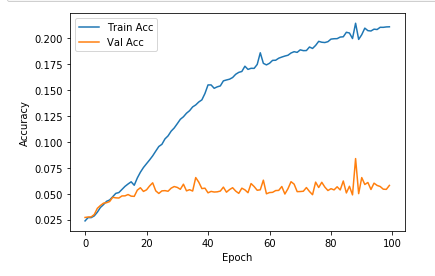
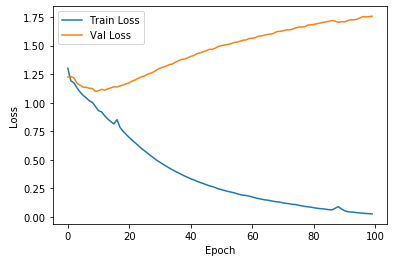
    Sample : 7000

    Batch Size : 64

    Max Cumulative Blue Score : 0.9

This model turned the translations pretty close to the target sentence. Also this model performed well on the validated text.





## WORD 2 WORD ATTENTION MODEL

Attention is one of the most influential ideas in the Deep Learning community. Even though this mechanism is now used in various problems like image captioning and others, it was initially designed in the context of Neural Machine Translation using Seq2Seq Models.

**Need for Attention Model:**

The seq2seq model is normally composed of an encoder-decoder architecture, where the encoder processes the input sequence and encodes/compresses/summarizes the information into a context vector (also called as the “thought vector”) of a fixed length. This representation is expected to be a good summary of the entire input sequence. The decoder is then initialized with this context vector, using which it starts generating the transformed output.

A critical and apparent disadvantage of this fixed-length context vector design is the incapability of the system to remember longer sequences. Often is has forgotten the earlier parts of the sequence once it has processed the entire the sequence. The attention mechanism was born to resolve this problem.

**Concept:**

For the illustrative purposes, let’s consider the example below:

Input (English) Sentence: “Clark is a good boy”

Target (Hindi) Sentence: “क्लार्क एक अच्छा लड़का है”

In this case, we are using a GRU layer instead of a LSTM layer. The reason being that LSTM has two internal states (hidden state and cell state) and GRU has only one internal state (hidden state). This was done in order to make the model less computationally intensive and thereby faster as well.

In the traditional Seq2Seq model, we discard all the intermediate states of the encoder and use only its final states (vector) to initialize the decoder. This technique works effectively for smaller sequences, however as the length of the sequence increases, a single vector becomes a bottleneck and it gets very difficult to summarize long sequences into a single vector. This observation was made empirically as it was noted that the performance of the system decreases drastically as the size of the sequence increases. The central idea behind Attention is not to throw away those intermediate encoder states but to utilize all the states in order to construct the context vectors required by the decoder to generate the output sequence.

As human beings we are quickly able to understand these mappings between different parts of the input sequence and corresponding parts of the output sequence. However it is not that straight forward for artificial neural network to automatically detect these mappings.

Thus, the Attention mechanism is developed to ***“learn”*** these mappings through Gradient Descent and Back-propagation.

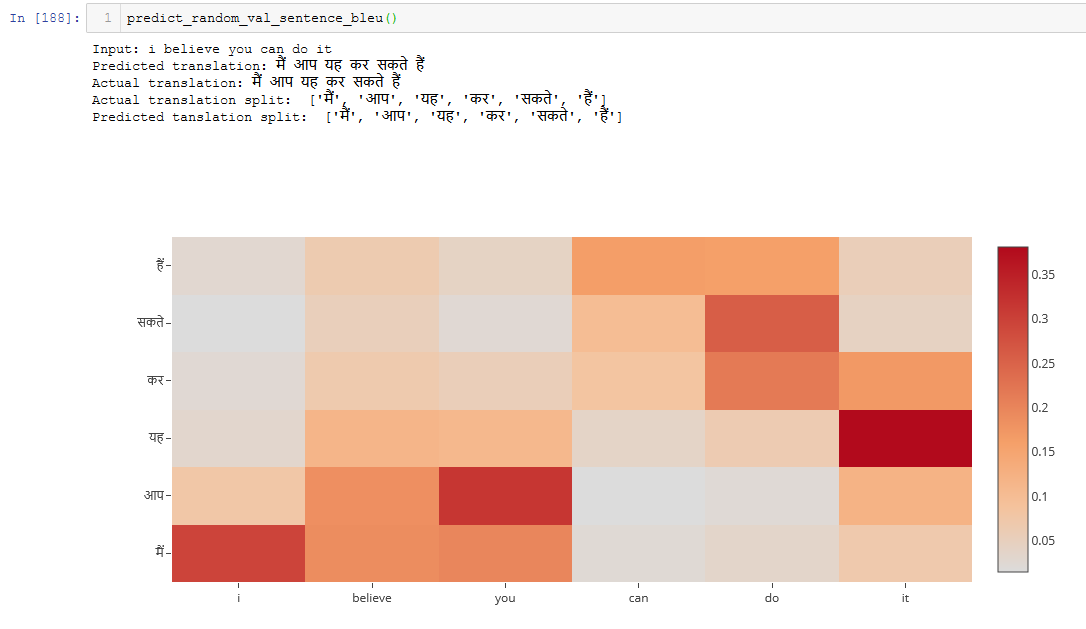
**Code Implementation:**

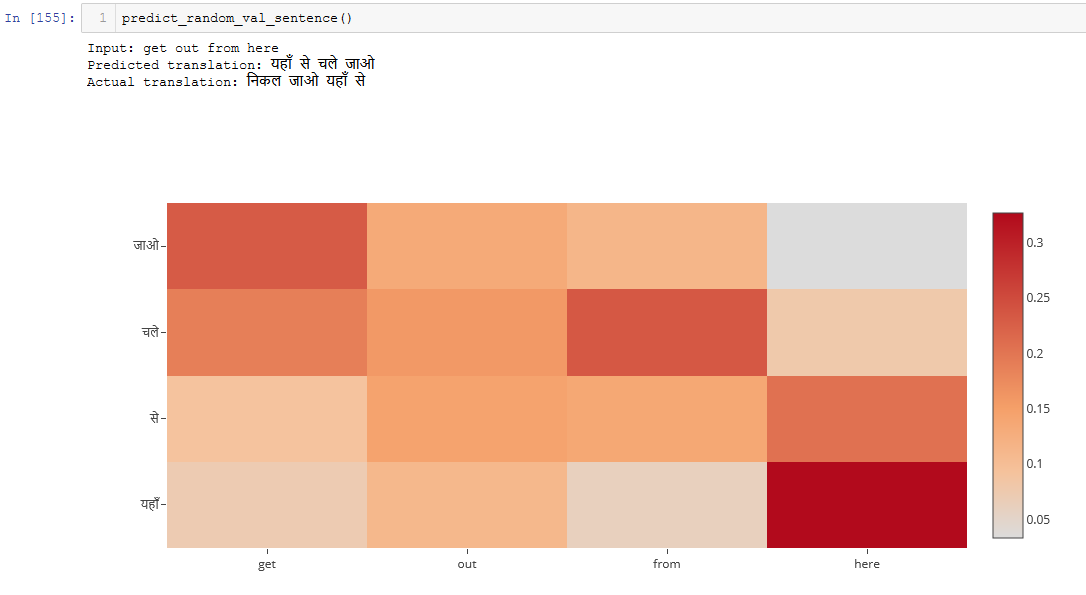
We perform the basic cleaning and preprocessing by removing all the numbers and punctuation from the raw data. Then, we create a class to map every word to an index and vice-versa for any given vocabulary. Now using the model sub-classing API of TensorFlow, we defined the model. We then defined the optimizer, loss function and checkpoints. After training the model for 12 epochs, we then performed inference setup and testing.

**Output Visualization:**

Notice that the cell at the intersection of “it” and “यह” is pretty dark. This means when the decoder predicts the word “यह”, it is paying more attention to the input word “it” (which is what we wanted).

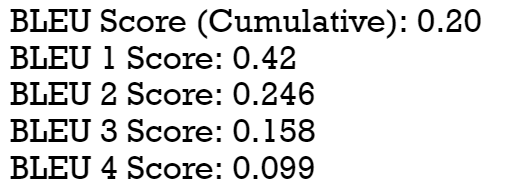
Similarly, while predicting the word “मैं”, the decoder pays a lot of attention to the input word “I”.





From the output, we can see that the model is able to find the correct local mappings between the input and the output sequences which do match with our intuition.

The BLEU scores for the attention model are as follows:



Given more data and with more hyper parameter tuning, the results and mappings will definitely improve by a good margin.

# RESULTS

|  |  |
| --- | --- |
| **Model** | **Best obtained Average BLEU Scores** |
| Char 2 Char LSTM | 0.17 |
| Word 2 Word dual LSTM stacked layers | 0.65 |
| Word to Word LSTM Hyper parameter tuning | 0.40 |
| Word to word GRU using Attention model | 0.20 |

It is noteworthy that out of total of ~75000 samples, we realized that the models are too big to run enough epochs within a timeframe repeatedly enough to compare various tweaks in hyperparameters. Hence, Model 1 to Model 3 are run on ~10% of the sample while the Attention model was run on the entire sample set.

It yielded the above results which hare the Best Average Cumulative BLEU 4-gram scores over all the variations in the respective models. We can say that Model 2 with dual LSTM encoder Decoders stacking, 512 latent dimensions, batch size 64, 7000 samples, Adam optimizer run over 100 epochs gave the all-round results of 0.65.

Attention model also gave a good performance metric over merely 12 epochs over the 48 hours that it ran.

# CONCLUSION & WAY FORWARD

After running variations in models and hyperparameters we observed that the best balance of training speed and performance was possible (BLEU score) through ADAM as the optimizer as it accommodates not just the learning rates, momentums but also prioritizes current set of rates over the previous ones so that the performances are more influenced in the recent context. Hence this would likely be our go to model for larger sample sizes.

Even though Deeper capacity Word to Word LSTM models present great results, Attention model though computationally expensive can give great performance over larger epochs. This can be done using “Transformer model” which uses only attention and gets rid of all Convolutional and Recurrent layers making it highly “parallelizable” and computationally efficient.

# REFERENCES

* <https://arxiv.org/abs/1409.3215>
* <https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346>
* <https://github.com/tensorflow/nmt>
* <https://arxiv.org/pdf/1409.0473.pdf>