CSYE 7245 **Big Data Sys & Int Analytics**

Language Translation Using LSTM and GRU

Under the guidance of

Prof. Nick Bear Brown

Chetan M Jadhav

Jadhav.ch@husky.neu.edu

NUID: 001836501

1. Abstract

In this research paper, we want to build language translation model that aims to translate Italian to English. Although, there are big players like Google that offers real time and accurate translation. In this paper we will concentrate on how the different types of neural networks which uses seq2seq model translates the language. Especially we will be using Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) with Attention. We will evaluate the model by Bilingual evaluation study (BLEU) and determine the performance of the model by checking the loss. The projects aim to tweak a designated number of parameters such as number of epochs, Activation function and the number of records to train the model to see how this will affect the accuracy also we are checking how the tweaking the above-mentioned parameters will affect the training time and the efficiency of the model. After running couple of models, the project concludes that the GRU with Attention comparatively performs better than other seq2seq models in the experiment.

2. Introduction

Recurrent Neural Networks (RNN) have revealed promising outcomes in several machine learning tasks, specifically when input or output are of flexible length. More recently its been stated that recurrent neural networks can accomplish well in building the machine translation model.

As we know that recurrent neural networks are doing well in machine translation field. We are more concerned in building a machine translation model using two closely related variants which are LSTM and GRU. These two variants of works well with seq2seq framework where the input is fed sequentially, and output is led out sequentially. The draw back which vanilla recurrent neural network had of vanishing gradient problem is resolved by using these two variants of RNN as these two can memorize more and can omit if the data is not needed in its memory and can work on longer sentences which was an issue using vanilla RNN.

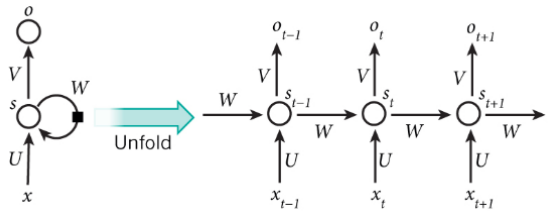
Based on our couple of experiments we can say that LSTM and GRU performs well in translating the language but GRU's efficiency is little bit more compared to that of LSTM, but training time is much more than that of LSTM.

3. Background

3.1 Recurrent Neural Network:

A recurrent neural network (RNN) is an extension of the conventional feedforward neural network, which can handle variable length sequence input. The core idea behind the RNN is to make use of sequential data. In traditional neural network we adopt that all the inputs are autonomous of each other. But many of tasks doesn't work this way and the traditional neural network was of no use when the model must forecast what's going to come next in the sentence as it needs to remember the previous couple of words, that's when the Recurrent Neural Network came to existence. This is called recurrent because of the network achieves the same task for every element of a sequence, with

output depending on the previous calculations. Further way of explaining the recurrent term is remembering the old computation in the memory.



Traditional Recurrent Neural Network

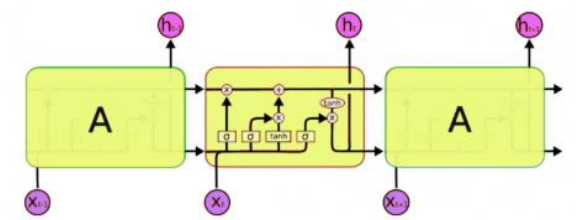
The above picture shows an RNN being unfolded into a full network where the network will be unfolded the times of the input. For example, if there is 3 words input then the RNN will unfold 3 times to train the model and to provide the output i.e., one layer for each word.

While using RNN, they came across the issue called vanishing gradients as the RNN is very old and this issue emerged as the major issue to recurrent neural network's performance. For recurrent neural networks we want to have extended memories so that system can join data relationships at significance at times but as we had more time steps, the more chance we have back-propogation gradients either hoarding and exploding or vanishing down to nothing.

The above obstacle of vanishing gradients is solved by the long short term memory and the gated recurrent unit, we will discuss about them in our next section.

3.2 Long Short-Term Memory (LSTM):

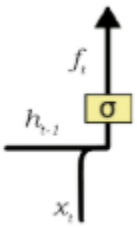
Long short term memory was developed as the solution to the traditional recurrent neural network and this work and have internal gates that regulates the data flow to the next layer. These gates can learn which data in a sequence is important to keep or to forget, by this it can keep only the data which is relevant and which is helpful in predecting the next word. And about the LSTM gates we going to discuss in brief in this section so we have better understanding on the same.



The typical LSTM network have different memory blocks which is termed by the name "Cells" and there are two states that are being shifted to the next cell called cell state and the hidden state. The memory blocks are accountable for remembering things and operations to this memory and is done through the mechanism called gates.

3.2.1 Forget Gate:

Forget gate is accountable for removing or throwing away the information that is not needed and this gate have two inputs h_{t-1} and x_t, h_{t-1} is the hidden state from the previus cell and x_t is the input of the current time step.

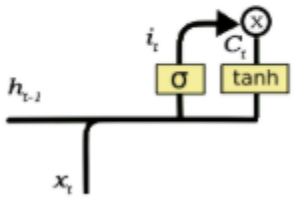


Both the inputs h_{t-1} and x_t are multipied before passing to the sigmoid function where the sigmoid function outputs vector of 0 to 1. Basically its sigmoid who decides whether to keep the data which is there in this gate or to throw away. For example, if the sigmoid output is 0 then it means the data which is

passed to the gate needs to be forgotten and if the value of sigmoid is 1 then the information is passed to the next layer.

3.2.2 Input Gate:

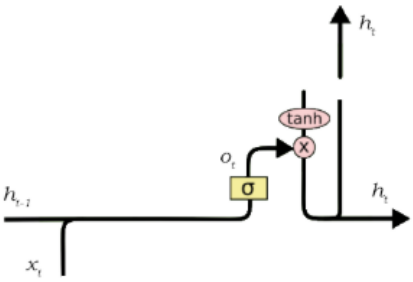
The input gate is accountable for addition of the information to the cell state and the process is expained with the diagram below



Input gate takes the two inputs hidden state from previous cell h_{t-1} and current input x_t. Both the inputs multiplied and passed it to the sigmoid and tanh function before again passing it to the multiplier where tanh function outputs in the range of -1 to 1 this value passes the useful information to the cell state.

3.2.3 Output Gate:

The output gate is accountable for the selecting the useful information from the current cell state and showing it as an output to the next layer.

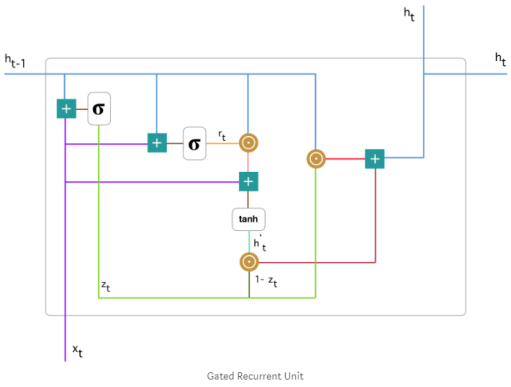


Output gate takes the two inputs hidden state from previous cell h_{t-1} and current input x_t. Both the inputs multiplied and passed it to the sigmoid function and passing it to the tanh function where the tanh function sacles the output in the range of -1 to 1 and multiplying the value of this filet to the vector created by tanh function will be the ouppput to the next layer which acts as h_{t-1} to the next layer.

LSTM's are very promising but its very difficult in training them and lot of computation is needed to train the model.

3.3 Gated Recurrent Unit:

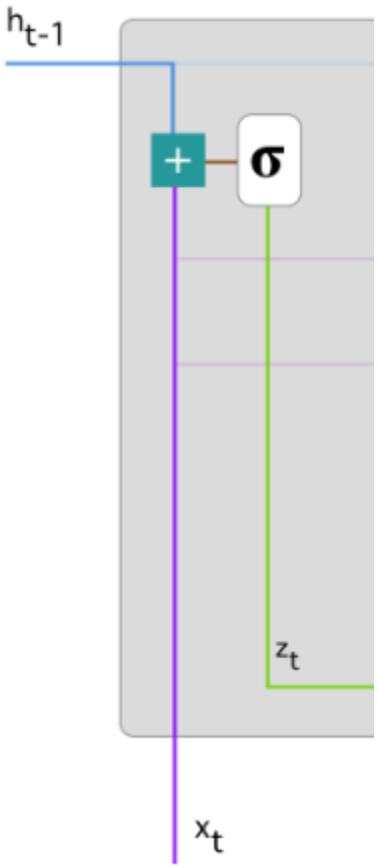
Gated Recurrent Unit was introduced by the K.Cho in 2014 and this is almost similar to LSTM and can be called enhanced LSTM. As LSTM even GRU has the gate concept and has 2 gates instead of 3 which we will discuss in detail in this section. Below is the diagram of the GRU.



And the notations are as mentioned below which will help to understand the equation

3.3.1 Update Gate:

The Update gate has two inputs hidden state from previous cell h_{t-1} and current input x_t. and this gate is accountable to check the piece of information and decide if it needs to update the cell



Both the inputs will be passed to the plus operation which again will be passed to the sigmoid function which vectors the output in the range of 0 to 1 and each inputs carries its own weight and can be defined by the below formula

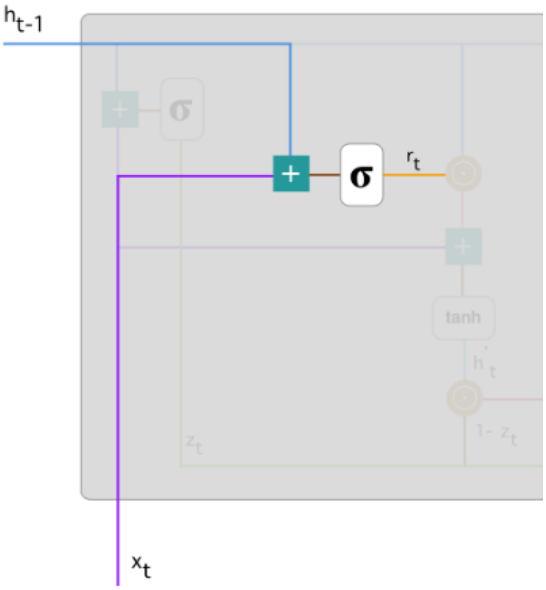
$Z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$

The input x_t is multiplied by its own weight $W^{(z)}$ and input h_{t-1} is mulitplied by its own weight $U^{(z)}$.

This update gate helps th ecell to determine how much of the information needs to be passed along and which information needs to be dropped and not to passed to the next layer.

3.3.2 Reset Gate:

Reset gate is accountable for deciding what piece information needs to be forgotten and reset gate also takes two inputs hidden state from previous cell h_{t-1} and current input x_t.



Both the inputs multiplied with their own weight and added together before passing it to the sigmoid function which vectors the output in the range of 0 to 1 and can be defined by the below formula

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$

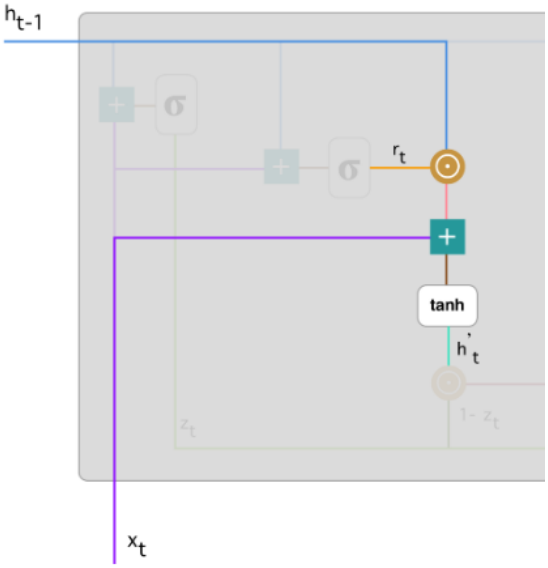
The output of the sigmoid function determines if the information is passed or forgotten. for example, if the output is 1 then the data is passed and if the output is 0 then the piece of information is dropped.

3.3.3 Current Memory Content:

By making use of the reset gate's output which will store the relevant information from the past and can be calculated by the below mentioned formula

$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

The input x_t is multiplied by its own weight W and h_{t-1} with its own weight U. and compute the Hadamard product between reset gate's output r_t and Uh_{t-1} and add it with the input Wx_t before passing it to the nonlinear tanh function which will vector the output in the range of -1 to 1

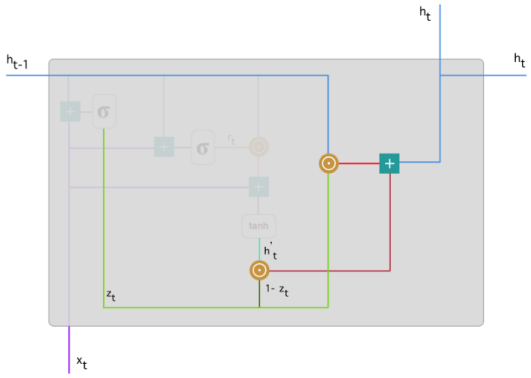


3.3.4 Final memory at current time step:

In this step, at the end it will hold the information which we pass to the next layer h_{t-1}, so to do this we require the update gate's output z_t and the output can be computed by the below formula

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t'$$

Here we will multiply by Hadamard product function the hidden input with the update gate's output and then Hadamard multiplier on $(1\text{-}z_t)$ and h_t before passing it to the plus operator.



This is how GRUs can store and update using reset and update gates which eliminates vanishing gradient problem.

4. Approach

After consideration of every aspect of all the models I decided to go with the LSTM and GRU with attention as the vanilla RNN have issues of vanishing gradient problem. We will be training the model by gated RNN models on the 18 MB dataset which has Italian and English words.

the project decided to build one model of LSTM by taking 50000 records from the dataset and training the model by splitting the data, in which 80% of the data is designated to the training and 20% of the data will be kept for testing. So, the data is vectorized and

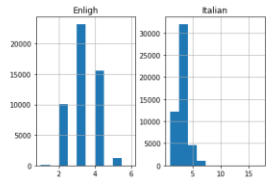
padded by making use of the Keras's function Tokenizer and pad_sequences.
And we will build models by making use of GRU with adding Attention vector to it. We will build the data first by taking 2000 records from the data and building the model by splitting the data in the ratio of 80 to 20% for training and testing purpose. And the model is trained for the 30 epochs and the same is checked for the couple of different number of records and by changing the epochs.
.

5. Code Implementation

After deciding on the approach, the model was built and below are the highlights of the LSTM model and later in this section we will go through the GRUs

- The data is first loaded to the jupyter notebook. Cleaned the data, Processed the data, converted the case, removed all the punctuation and later split the data into Italian and English and store the data in two different variables
- The split data is vectorized and ran through pad_sequence which will pad the sequence and if the length of the input and adds zero when the length of the input is less than the defined length
- The length of the data is checked

```
length_dframe = pd.DataFrame({'Enligh':eng_length, 'Italian':ita_length})
length_dframe.hist(bins = 10)
plt.show()
```



- The model is built using the function and compiled

```
def build_model(ItalianVocabSize, EnglishVocabSize, ItalianLength, EnglishLength, Units):
    model = Sequential()
    model.add(Embedding(ItalianVocabSize, Units, input_length=ItalianLength, mask_zero=True))
    model.add(LSTM(Units))
    model.add(RepeatVector(EnglishLength))
    model.add(LSTM(Units, return_sequences=True))
    model.add(Dense(EnglishVocabSize, activation='softmax'))
    model.summary()
    return model
```

- Training the model

```
filename = 'SavedModel_Checkpoint'
checkpoint = ModelCheckpoint(filename, monitor='val_loss', verbose=1, save_best_only=True, mode='min')

history = model.fit(x_train, y_train.reshape(y_train.shape[0], y_train.shape[1], 1),
          epochs=35, batch_size=512,
          validation_split = 0.2,
          callbacks=[checkpoint], verbose=1)
```

```
WARNING:tensorflow:From C:\Users\cheta\Anaconda3\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
WARNING:tensorflow:From C:\Users\cheta\Anaconda3\lib\site-packages\tensorflow\python\ops\math_grad.py:102: div (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
Train on 32000 samples, validate on 8000 samples
Epoch 1/35
32000/32000 [==============================] - 15s 464us/step - loss: 4.3675 - val_loss: 3.7549

Epoch 00001: val_loss improved from inf to 3.75490, saving model to SavedModel_Checkpoint
Epoch 2/35
32000/32000 [==============================] - 10s 318us/step - loss: 3.6031 - val_loss: 3.5102
```

- Prediction

```
pred_dframe.head(50)
```

	Italian	actual	predicted
0	ora sei al sicuro	now youre safe	now youre safe
1	sono sempre pronta	im always ready	im always ready
2	ti amavo	i loved you	i loved you
3	tom lha visto	tom saw it	did tom see you
4	lui non ha detto niente	he said nothing	he said nothing
5	qualcuno lha visto	somebody saw you	somebody saw see
6	tom sta parlando	tom is talking	tom is speaking
7	state a guardare	stay and watch	stay and watch
8	spegnetelo	switch it off	turn that off
9	sono sistematico	im methodical	im resourceful
10	è eccitata	are you excited	are you excited
11	ero impegnata	i was busy	i was busy
12	io vedo il cane	i see the dog	i see the dog

Now about the GRU's code implementation, the initialization like cleaning, processing, tokenizing and Pad_sequence is done almost the same way as done in the LSTM model.

- Building GRU function, Encoder, Decoder function
GRU Function

```
def gru(units):
    return tf.keras.layers.GRU(units,
                               return_sequences=True,
                               return_state=True,
                               recurrent_activation='sigmoid',
                               recurrent_initializer='glorot_uniform')
```

Encoder

```
class Encoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
        super(Encoder, self).__init__()
        self.batch_sz = batch_sz
        self.enc_units = enc_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = gru(self.enc_units)

    def call(self, x, hidden):
        x = self.embedding(x)
        output, state = self.gru(x, initial_state = hidden)
        return output, state

    def initialize_hidden_state(self):
        return tf.zeros((self.batch_sz, self.enc_units))
```

Decoder

```
class Decoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
        super(Decoder, self).__init__()
        self.batch_sz = batch_sz
        self.dec_units = dec_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = gru(self.dec_units)
        self.fc = tf.keras.layers.Dense(vocab_size)

        # used for attention
        self.W1 = tf.keras.layers.Dense(self.dec_units)
        self.W2 = tf.keras.layers.Dense(self.dec_units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, x, hidden, enc_output):
        # enc_output shape == (batch_size, max_length, hidden_size)

        # hidden shape == (batch_size, hidden size)
        # hidden_with_time_axis shape == (batch_size, 1, hidden size)
        # we are doing this to perform addition to calculate the score
        hidden_with_time_axis = tf.expand_dims(hidden, 1)

        # score shape == (batch_size, max_length, 1)
        # we get 1 at the last axis because we are applying tanh(FC(EO) + FC(H)) to self.V
        score = self.V(tf.nn.tanh(self.W1(enc_output) + self.W2(hidden_with_time_axis)))

        # attention_weights shape == (batch_size, max_length, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * enc_output
        context_vector = tf.reduce_sum(context_vector, axis=1)

        # x shape after passing through embedding == (batch_size, 1, embedding_dim)
        x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
        output, state = self.gru(x)

        # output shape == (batch_size * 1, hidden_size)
        output = tf.reshape(output, (-1, output.shape[2]))

        # output shape == (batch_size * 1, vocab)
        x = self.fc(output)

        return x, state, attention_weights
```

- Training the model

```
EPOCHS = 30

for epoch in range(EPOCHS):
    start = time.time()

    hidden = encoder.initialize_hidden_state()
    total_loss = 0

    for (batch, (inp, targ)) in enumerate(dataset):
        loss = 0

        with tf.GradientTape() as tape:
            enc_output, enc_hidden = encoder(inp, hidden)

            dec_hidden = enc_hidden

            dec_input = tf.expand_dims([targ_lang.word2idx['<start>']] * BATCH_SIZE, 1)

            # Teacher forcing - feeding the target as the next input
            for t in range(1, targ.shape[1]):
                # passing enc_output to the decoder
                predictions, dec_hidden, _ = decoder(dec_input, dec_hidden, enc_output)

                loss += loss_function(targ[:, t], predictions)

                # using teacher forcing
                dec_input = tf.expand_dims(targ[:, t], 1)

        batch_loss = (loss / int(targ.shape[1]))

        total_loss += batch_loss

        variables = encoder.variables + decoder.variables

        gradients = tape.gradient(loss, variables)

        optimizer.apply_gradients(zip(gradients, variables))

        if batch % 100 == 0:
            print('Epoch {} Batch {} Loss {:.4f}'.format(epoch + 1,
                                                         batch,
                                                         batch_loss.numpy()))

    print('Epoch {} Loss {:.4f}'.format(epoch + 1,
                                        total_loss / N_BATCH))
    print('Time taken for 1 epoch {} sec\n'.format(time.time() - start))
```

```
Epoch 2 Batch 0 Loss 2.3634
Epoch 2 Loss 2.1523
Time taken for 1 epoch 48.138915538787842 sec

Epoch 3 Batch 0 Loss 1.9294
Epoch 3 Loss 1.8879
Time taken for 1 epoch 48.516262054443362 sec
```

- Prediction

```
test = 'corri'
translate(sentence=test, encoder=encoder, decoder=decoder
```

```
Input: <start> corri <end>
Predicted translation: run <end>
```

6.Discussion

6.1 Conclusion

The built model was able to translate the Italian words to the English most of the time and by this research I can conclude that GRU with attention was able to translate comparatively better than the other model, but the training time was high for the GRU.

6.2 Evaluation

The evaluation of the experiment was done on two methods first we checked the BLEU score which stands for the Bilingual Evaluation study and determine the performance of the model by checking the loss. Also, the evaluation is also done on training time of the model

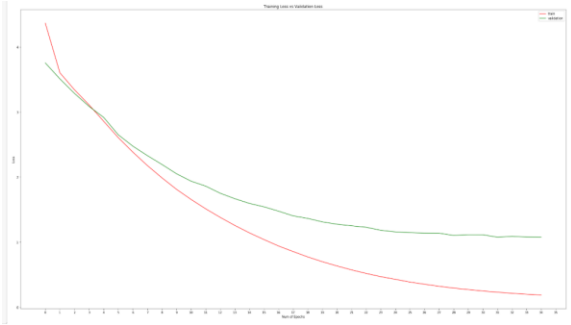
6.3 Result

BLEU Score:

```
evaluate_model(model, English_tokenizer, x_train[:50,:], train[:50,:])
```

```
src=[è per lei], target=[thats for you], predicted=[thats for you]
src=[tom si è addormentato], target=[tom fell asleep], predicted=[tom fell asleep]
src=[è di fretta], target=[are you in a rush], predicted=[are you in a rush]
src=[loro sono miei], target=[theyre mine], predicted=[theyre mine]
src=[loro lo amano], target=[they love that], predicted=[they love that]
src=[noi vogliamo degli impieghi], target=[we want jobs], predicted=[we want jobs]
src=[siete energiche], target=[are you proactive], predicted=[youre energetic]
src=[tom lavora qui], target=[tom works here], predicted=[tom works here]
src=[mi dia una scelta], target=[give me a choice], predicted=[give me a choice]
src=[tom fiutò], target=[tom sniffed], predicted=[tom sniffed]
BLEU: 0.114642
```

Loss:



The training time of the models have been tabulated below.

Model Name	Record Count	Epochs	Training Time	Loss
LSTM	50,000	35	143 secs	0.1853
	100,000	60	613 secs	0.0975
GRU	2,000	30	90 secs	0.1649
	4,000	40	360 secs	0.0551
	6,000	60	840 secs	0.046

6.3 Future work

The future work on this will be translating from any language to any language with more efficiency and to play around other architectures which can translate the language much better.

7. References

1. Machine Learning is Fun Part 5: Language Translation with Deep Learning and the Magic of Sequences https://medium.com/@ageitgey/machine-learning-is-fun-part-5-language-translation-with-deep-learning-and-the-magic-of-sequences-2ace0acca0aa
2. Microsoft's Language Translation AI has Reached Human Levels of Accuracy https://www.analyticsvidhya.com/blog/2018/03/microsofts-claims-language-translation-ai-reached-human-levels-accuracy/
3. Machine Learning Translation and the Google Translate Algorithm https://blog.statsbot.co/machine-learning-translation-96f0ed8f19e4
4. https://www.youtube.com/watch?v=nRBnh4qbPHI&vl=en
5. http://www.manythings.org/anki/
6. A Gentle Introduction to Neural Machine Translation https://machinelearningmastery.com/introduction-neural-machine-translation/
7. How to Develop a Neural Machine Translation System from Scratch https://machinelearningmastery.com/develop-neural-machine-translation-system-keras/
8. https://www.youtube.com/watch?v=vI2Y3I-JI2Q
9. Neural Machine Translator with Less than 50 lines of Code + Guide https://towardsdatascience.com/neural-machine-translator-with-less-than-50-lines-of-code-guide-1fe4fdfe6292
10. Neural Machine Translation with Attention https://www.tensorflow.org/alpha/tutorials/text/nmt_with_attention
11. How to Tune LSTM Hyperparameters with Keras for Time Series Forecasting https://machinelearningmastery.com/tune-lstm-hyperparameters-keras-time-series-forecasting/
12. Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
13. Keras LSTM tutorial – How to easily build a powerful deep learning language model https://adventuresinmachinelearning.com/keras-lstm-tutorial/
14. Deep Learning Basics: Gated recurrent unit (GRU) https://medium.com/mlrecipies/deep-learning-basics-gated-recurrent-unit-gru-1d8e9fae7280

15. What is a Recurrent Neural Networks (RNNS) and Gated Recurrent Unit (GRUS) https://towardsdatascience.com/what-is-a-recurrent-nns-and-gated-recurrent-unit-grus-ea71d2a05a69
16. https://en.wikipedia.org/wiki/Recurrent_neural_network

8. Appendix

Github link:

https://github.com/chetanmj23/BDI-Final-Project