COMPARISON OF NMT MODELS

MACHINE TRANSLATION FROM ENGLISH TO HINDI

Faculty Mentor:

Lyla B.Das Ma'am
Associate Professor,
ECD
NITC

Prepared and Performed BY:

Naveen Ambarish M.
(5th sem. NITT)
Inst, Roll no. : 107119077

Saurabh Patel
(5th sem NITRR)
Inst, Roll no. : 19116081

Abstract:

In NMT cyclic neural networks, especially the LSTMs, RNNs and GRUs have been regarded as the latest methods in the field of neural machine translation. When the neural networks are running the sequence of sentences from one language to another are processed one by one uniquely with each word in English language is tagged with the corresponding word in the Hindi Language. With the advancement in the field of neural machine translation, transformer models were developed using self-attention concept which shows better accuracy than the simple NMT done with attention models. Our project in transformer shows a high BELU score of 40.48 than simple NMT with a BELU score 18.5.

Introduction:

To solve different problems that arise in our day-to-day life, people across the world should be able to discuss with each other to come with unique and more accurate solutions. But the different language present across the world with different words becomes the bottleneck. Machine translation using computers are done automatically once they are being trained with the models and are relatively cheaper compared to the human translators. Machine translation has developed due to the advancement of technology made in natural language processing. The emergence of the neural machine translation has significantly improved the quality of the machine translation.

The Neural Machine translation that was presently used are based on the concept of encoder and decoder. This model uses the two neural networks of encoder and decoder separately. The encoder is used to read the source sentence and the decoder part is used to decode the vector from the source sentence and generates the corresponding sentence in the target language.

The simple NMT faces a bottleneck which is nothing but when the encoder reads a long input sentence in source language and then when encoding it into the thought vector which is nothing, but the last hidden unit of the RNN or LSTM. So, the initial word embeddings of the encoder get ignored. So, to avoid this attention models were developed which assigns attention weight to all the words in the encoder and the most important words are given with some attention.

Finally, the recent advancement in natural language processing is transformers. It is based on self-attention models. In our project we have done with the auto encoder which was a pretrained model developed by the Facebook. It gives the highest accuracy among the simple NMT and attention models.

Related Work:

NMT structure proposed by Sutskever et.al whose architecture is based on basic sequence to sequence but does not use attention mechanism. It uses the simple technique of reversing the input sequence because the first input sequence is directly dependent on first target sequence. Since the words in target language are directly influenced by the words in the input language nearer to it. When the input sequence is trained in the correct order, then the first word in the output language cannot remember the first word in input language due to long distance between them.

NMT structure designed by Bahdanau et al, that uses the attention model in the basis sequence to sequence network. In this model the RNN was completely replaced by GRU and LSTM. Also, this system uses the Bidirectional RNN in encoder part. This model incorporates two hidden state vectors which are concatenated to give the context vector.

Multilingual Denoising Pre-training for Neural Machine Translation by Yinhan Liu et al (Facebook AI) : This paper demonstrated that multilingual denoising pre-training produces significant performance gains across a wide variety of machine translation tasks and presented mBART as a Seq2Seq auto-encoder pre-trained on large scale corpora in many languages.

Model:

Text Pre-processing:

Any data available cannot be trained directly or translated. It may contain some unimportant words like digits, punctuations etc. or some unknown words other than the source or target language. So, the data needs to be pre-processed to make it ready for training the model. Also, the computer cannot understand text sentences, so these should be mapped with some unique numbers for each unique words present in the source and target language.

(i) Simple Neural Machine Translation (NMT):

It was designed using simple encoder and decoders with LSTM layer. Before the input is taken into the LSTM layer it needs to be converted to word embeddings. The vector size of these word embeddings is given as the input by the user.The mapped unique integer values from the unique words in the corpus is given trained as to get a unique embedding vector for each word. The input source vector is being reversed before training which is being mentioned above. This model was made to run about 25 epochs in our project to avoid over-fitting problem.

(ii) Neural Machine Translation (NMT) using Attention Layer:

This was being constructed same as that of the NMT. But the difference was that it was constructed using the bidirectional LSTM layer in the encoder. So, the encoder layer contains both the forward as well as backward states which are concatenated together as an array or list and then trained as encoder states. Here, the states are nothing, but the hidden state value and the cell memory value of the present state is stored to predict the next word.

Attention model scores each context vector G_i with respect to the current hidden state vector in Decoder layer Z_{t-1} using the attention score e_i^t using the following model:

$$e_i^t = f_{ATT}(Z_{t-1}, G_i, \{\alpha_j^{t-1}\}_{j=1}^{L_{in}})$$

Where,

the attention weights α_i^t using the equation given below:

$$\alpha_i^t = \frac{e_i^t}{\sum_{j=1}^{L_{in}} e_j^t}$$

The attention weights are then used to compute the new context vector:

$$B_t = \psi(\{G_i\}_{i=1}^{L_{in}}, \{\alpha_i^t\}_{i=1}^{L_{in}})$$

Then this context vector $B_t$is used to update the hidden state and output of the decoder which in case of an RNN-Language Model is as follows:

$$Z_t = f(WZ_{t-1} + UY_{t-1} + SB_t)$$

$$Y_t = h(VZ_t + U'Y_{t-1} + S'B_t)$$

The attention score in our project is calculated as follows:

$$e_i^t = V_a^T \tanh(W_a Z_{t-1} + U_a G_i)$$

This NMT with attention model also were made to train for 25 epochs as mentioned in the previous case.

(iii) Neural Machine Translation (NMT) using Transformers:

The Transformer was originally introduced in "Attention is All you Need" paper. It is basically a novel neural network architecture which is based on self-attention mechanism well suited for language understanding.

Transformer outperforms both recurrent and conventional models on neural machine translation in terms of the higher translation quality. Adding to this, it requires less computation on training and proves to be a much better fit for modern machine learning hardware.

As compared to the sequential RNN architecture, Transformer performs a constant, small number of steps. In each step, self-attention mechanism is applied which performs the task of modelling relationships between all words in a sentence irrespective of their positions.

To compute the next representation of a given word, the Transformer compares it to every other word in the sentence. This results in an attention score for all other words which determine how much each of them will contribute to the next representation of that given word. These attention scores are further used as weights to calculate the weighted average for all words' representations which is then fed to a fully connected network to generate a new representation of the word.

The Transformer starts by generating embeddings for each word, all fed at once. By using self-attention, it accumulates information from all the other words generating a new representation per word informed by the entire context. There is an iteration of this step in parallel for all the words successively generating new representations.

The operation of decoder remains the same except for generating one word at a time from left to right. In addition to other previously generated words, it also attends to the final representations generated by encoder.

In this project, we have used the *mBART* Transformer model developed by Facebook AI research. It is the first method for pre-training a complete Seq2Seq model on large scale monolingual corpora in many languages using the BART objective (Lewis et al., 2019) by denoising full texts in multiple languages. On the contrary, the previous approaches have focused only on the encoder, decoder, or reconstructing parts of the text.

Input texts are noised by phase masking and sentence permuting. A single Transformer (Vaswani et al., 2017) model further learns to recover the texts. mBART pre-trains a complete autoregressive Seq2Seq model. mBART is trained once for all languages, providing a set of parameters that can be directly fine-tuned for any of the language pairs for both supervised and unsupervised machine translation without any task-specific modifications.

Codes :-

GitHub: code

(i) Simple NMT :

```
!pip install sacrebleu
import pandas as pd
import numpy as np
import tensorflow as tf
import string
import re
import math
import os
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding
from tensorflow.keras.optimizers import RMSprop
from sacrebleu import sentence_bleu
df=pd.read_csv("/content/drive/MyDrive/hin.txt", sep='\t', header=None,
names=["english_sentence","hindi_sentence","path"])
print(df.head(10))
df= df.drop(columns=['path'])
print(df.shape)
print(df.isnull().sum())
df['english_sentence']=df['english_sentence'].apply(lambda x: x.lower())
df['hindi_sentence']=df['hindi_sentence'].apply(lambda x: x.lower())
df['english_sentence']=df['english_sentence'].apply(lambda x: x.strip())
df['hindi_sentence']=df['hindi_sentence'].apply(lambda x: x.strip())
df['english_sentence']=df['english_sentence'].apply(lambda x: re.sub(" +", " ", x))
df['hindi_sentence']=df['hindi_sentence'].apply(lambda x: re.sub(" +", " ", x))
df['english_sentence']=df['english_sentence'].apply(lambda x: ".join(ch for ch in x
if ch not in string.punctuation))
df['hindi_sentence']=df['hindi_sentence'].apply(lambda x: ".join(ch for ch in x if
ch not in string.punctuation))
df['english_sentence']=df['english_sentence'].str.replace('\d+', '')
df['hindi_sentence']=df['hindi_sentence'].str.replace('\d+', '')
start = '<s> '
end = ' </s>'
df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x : start + x + end)
english_vocab = {}
for i in df.english_sentence:
  for word in i.split():
    if word not in english_vocab:
english_vocab[word] = 1
    else:
english_vocab[word]+=1
```

```
hindi_vocab={}
for j in df.hindi_sentence:
  for a in j.split():
    if a not in hindi_vocab:
hindi_vocab[a] = 1
    else:
hindi_vocab[a]+=1
num_encoder_tokens=len(english_vocab.keys())
num_decoder_token=len(hindi_vocab.keys())
length = []
for i in df.english_sentence:
length.append(len(i.split(' ')))
max_input_length = max(length)
print('max_input_length: ', max_input_length)
length = []
for i in df.hindi_sentence:
length.append(len(i.split(' ')))
max_output_length = max(length)
print('max_output_length: ', max_output_length)
input_words = sorted(list(english_vocab.keys()))
target_words = sorted(list(hindi_vocab.keys()))
input_token_index = dict([(word, i) for i, word in enumerate(input_words)])
target_token_index = dict([(word, i) for i, word in enumerate(target_words)])
encoder_input_data = np.zeros((len(df.english_sentence), max_input_length),
dtype='float32')
decoder_input_data = np.zeros((len(df.hindi_sentence), max_output_length),
dtype='float32')
decoder_target_data = np.zeros((len(df.hindi_sentence), max_output_length,
num_decoder_token))
for i,(input_text, output_text) in enumerate(zip(df.english_sentence,
df.hindi_sentence)):
  for t, word in enumerate(input_text.split()):
encoder_input_data[i,t] = input_token_index[word]
  for t,word in enumerate(output_text.split()):
decoder_input_data[i,t] = target_token_index[word]
    if t > 0:
decoder_target_data[i,t-1,target_token_index[word]] = 1
latent_dim=300
encoder_inputs = Input(shape=(None,))
enc_emb=  Embedding(num_encoder_tokens, latent_dim, mask_zero =
True)(encoder_inputs)
encoder_lstm = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(enc_emb)
encoder_states = [state_h, state_c]
```

```
decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(num_decoder_token, latent_dim, mask_zero =
True)
dec_emb = dec_emb_layer(decoder_inputs)
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(dec_emb,
initial_state=encoder_states)
decoder_dense = Dense(num_decoder_token, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',metrics=['accuracy'])
model.summary()
model.fit([encoder_input_data, decoder_input_data], decoder_target_data,
batch_size=100, epochs=25)
encoder_model = Model(encoder_inputs, encoder_states)
encoder_model.summary()
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]

dec_emb2= dec_emb_layer(decoder_inputs)
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2,
initial_state=decoder_states_inputs)
decoder_states2 = [state_h2, state_c2]
decoder_outputs2 = decoder_dense(decoder_outputs2)

decoder_model = Model(
   [decoder_inputs] + decoder_states_inputs,
   [decoder_outputs2] + decoder_states2)
decoder_model.summary()
reverse_input_char_index = dict((i,char) for char, i in input_token_index.items())
reverse_target_char_index = dict((i,char) for char, i in target_token_index.items())


def decode_sequence(input_seq):
states_value = encoder_model.predict(input_seq)
target_seq = np.zeros((1,1))
target_seq[0, 0] = target_token_index['<s>']
stop_condition = False
decoded_sentence = ''
   while not stop_condition:
output_tokens, h, c = decoder_model.predict([target_seq] + states_value)
sampled_token_index = np.argmax(output_tokens[0, -1, :])
sampled_char = reverse_target_char_index[sampled_token_index]
```

```
decoded_sentence += ' '+sampled_char
      if (sampled_char == '</s>' or
len(decoded_sentence) > 50):
stop_condition = True
target_seq = np.zeros((1,1))
target_seq[0, 0] = sampled_token_index
states_value = [h, c]

   return decoded_sentence

k=2000
english =df.english_sentence[k:k+1].values[0]
actual = df.hindi_sentence[k:k+1].values[0]
predicted = decode_sequence(encoder_input_data[k:k+1])
print("The actual english sentence is:",english)
print("The actual hindi sentence is:",actual)
print("The predicted hindi sentence is:",predicted)
print("The BLEU score is :",sentence_bleu(predicted,[actual]).score)


k=52
english =df.english_sentence[k:k+1].values[0]
actual = df.hindi_sentence[k:k+1].values[0]
predicted = decode_sequence(encoder_input_data[k:k+1])
print("The actual english sentence is:",english)
print("The actual hindi sentence is:",actual)
print("The predicted hindi sentence is:",predicted)
print("The BLEU score is :",sentence_bleu(predicted,[actual]).score)
test_sentence = 'birds eat'
encoder_test_data = np.zeros((len(df.english_sentence), max_input_length),
dtype='float32')
for t, word in enumerate(test_sentence.split()):
encoder_test_data[1,t] = input_token_index[word]
predicted=decode_sequence(encoder_test_data[1:2])
correct='<s>पंछी खाना।</s>'
print("The  english sentence is:",test_sentence)
print("The predicted hindi sentence is:",predicted)
print("The BLEU score is :",sentence_bleu(predicted,[correct]).score)
```

(ii) NMT with attention:

```
import pandas as pd
import numpy as np
import tensorflow as tf
import string
import re
import math
import os
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding,
Bidirectional, Concatenate
from tensorflow.keras.optimizers import RMSprop
from sacrebleu import sentence_bleu
df=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Machine
Translation/hin.txt", sep='\t', header=None,
names=["english_sentence","hindi_sentence","path"])
df.head()
df=df.drop(columns=['path'])
df.shape
df.isnull().sum()
df['english_sentence']=df['english_sentence'].apply(lambda x: x.lower())
df['hindi_sentence']=df['hindi_sentence'].apply(lambda x: x.lower())
df['english_sentence']=df['english_sentence'].apply(lambda x: x.strip())
df['hindi_sentence']=df['hindi_sentence'].apply(lambda x: x.strip())
df['english_sentence']=df['english_sentence'].apply(lambda x: re.sub(" +", " ", x))
df['hindi_sentence']=df['hindi_sentence'].apply(lambda x: re.sub(" +", " ", x))
df['english_sentence']=df['english_sentence'].apply(lambda x: ''.join(ch for ch in x
if ch not in string.punctuation))
df['hindi_sentence']=df['hindi_sentence'].apply(lambda x: ''.join(ch for ch in x if
ch not in string.punctuation))
df['english_sentence']=df['english_sentence'].apply(lambda x: re.sub(r'\d+', '', x))
df['hindi_sentence']=df['hindi_sentence'].apply(lambda x:  re.sub(r'\d+','',x))
start = '<s> '
end = ' </s>'
df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x : start + x + end)
english_vocab = {}
for i in df.english_sentence:
  for word in i.split():
    if word not in english_vocab:
english_vocab[word] = 1
    else:
english_vocab[word]+=1

hindi_vocab={}
for j in df.hindi_sentence:
```

```
  for a in j.split():
    if a not in hindi_vocab:
hindi_vocab[a] = 1
    else:
hindi_vocab[a]+=1
num_encoder_tokens=len(english_vocab.keys())
num_decoder_token=len(hindi_vocab.keys())
length = []
for i in df.english_sentence:
length.append(len(i.split(' ')))
max_input_length = max(length)
print('max_input_length: ', max_input_length)
length = []
for i in df.hindi_sentence:
length.append(len(i.split(' ')))
max_output_length = max(length)
print('max_output_length: ', max_output_length)
input_words = sorted(list(english_vocab.keys()))
target_words = sorted(list(hindi_vocab.keys()))
input_token_index = dict([(word, i) for i, word in enumerate(input_words)])
target_token_index = dict([(word, i) for i, word in enumerate(target_words)])
encoder_input_data = np.zeros((len(df.english_sentence), max_input_length),
dtype='float32')
decoder_input_data = np.zeros((len(df.hindi_sentence), max_output_length),
dtype='float32')
decoder_target_data = np.zeros((len(df.hindi_sentence), max_output_length,
num_decoder_token))
for i,(input_text, output_text) in enumerate(zip(df.english_sentence,
df.hindi_sentence)):
  for t, word in enumerate(input_text.split()):
encoder_input_data[i,t] = input_token_index[word]
  for t,word in enumerate(output_text.split()):
decoder_input_data[i,t] = target_token_index[word]
    if t > 0:
decoder_target_data[i,t-1,target_token_index[word]] = 1
embedding_dim = 256
units = 1024


encoder_inputs = Input(shape=(max_input_length,))
enc_emb=  Embedding(num_encoder_tokens, embedding_dim)(encoder_inputs)
encoder_lstm = Bidirectional(LSTM(units=units//2,return_state=True,
return_sequences=True,recurrent_initializer='glorot_uniform'))
encoder_outputs, forward_state_h, forward_state_c, backward_state_h,
backward_state_c = encoder_lstm(enc_emb)
final_state_h = Concatenate()([forward_state_h,backward_state_h])
```

```
final_state_c = Concatenate()([forward_state_c,backward_state_c])
encoder_states = [final_state_h, final_state_c]
from tensorflow.python.keras.layers import Layer
from tensorflow.python.keras import backend as K
class AttentionLayer(Layer):
    """
    This class implements Bahdanau attention
(https://arxiv.org/pdf/1409.0473.pdf).
    There are three sets of weights introduced W_a, U_a, and V_a
     """

    def __init__(self, **kwargs):
super(AttentionLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        assert isinstance(input_shape, list)
        # Create a trainable weight variable for this layer.

self.W_a = self.add_weight(name='W_a',
                           shape=tf.TensorShape((input_shape[0][2],
input_shape[0][2])),
                           initializer='uniform',
                           trainable=True)
self.U_a = self.add_weight(name='U_a',
                           shape=tf.TensorShape((input_shape[1][2],
input_shape[0][2])),
                           initializer='uniform',
                           trainable=True)
self.V_a = self.add_weight(name='V_a',
                           shape=tf.TensorShape((input_shape[0][2], 1)),
                           initializer='uniform',
                           trainable=True)

super(AttentionLayer, self).build(input_shape)  # Be sure to call this at the end

    def call(self, inputs, verbose=False):
        """
        inputs: [encoder_output_sequence, decoder_output_sequence]
        """
        assert type(inputs) == list
encoder_out_seq, decoder_out_seq = inputs
        if verbose:
            print('encoder_out_seq>', encoder_out_seq.shape)
            print('decoder_out_seq>', decoder_out_seq.shape)

        def energy_step(inputs, states):
```

```
        """ Step function for computing energy for a single decoder state """

assert_msg = "States must be a list. However states {} is of type {}".format(states,
type(states))
        assert isinstance(states, list) or isinstance(states, tuple), assert_msg

        """ Some parameters required for shaping tensors"""
en_seq_len, en_hidden = encoder_out_seq.shape[1], encoder_out_seq.shape[2]
de_hidden = inputs.shape[-1]

        """ Computing S.Wa where S=[s0, s1, ..., si]"""
        # <= batch_size*en_seq_len, latent_dim
reshaped_enc_outputs = K.reshape(encoder_out_seq, (-1, en_hidden))
        # <= batch_size*en_seq_len, latent_dim
W_a_dot_s = K.reshape(K.dot(reshaped_enc_outputs, self.W_a), (-1, en_seq_len,
en_hidden))
        if verbose:
            print('wa.s>',W_a_dot_s.shape)

        """ Computing hj.Ua """
U_a_dot_h = K.expand_dims(K.dot(inputs, self.U_a), 1)  # <= batch_size, 1,
latent_dim
        if verbose:
            print('Ua.h>',U_a_dot_h.shape)
        """ tanh(S.Wa + hj.Ua) """
        # <= batch_size*en_seq_len, latent_dim
reshaped_Ws_plus_Uh = K.tanh(K.reshape(W_a_dot_s + U_a_dot_h, (-1,
en_hidden)))
        if verbose:
            print('Ws+Uh>', reshaped_Ws_plus_Uh.shape)

        """ softmax(va.tanh(S.Wa + hj.Ua)) """
        # <= batch_size, en_seq_len
e_i = K.reshape(K.dot(reshaped_Ws_plus_Uh, self.V_a), (-1, en_seq_len))
        # <= batch_size, en_seq_len
e_i = K.softmax(e_i)

        if verbose:
            print('ei>', e_i.shape)

        return e_i, [e_i]

    def context_step(inputs, states):
        """ Step function for computing ci using ei """
        # <= batch_size, hidden_size
c_i = K.sum(encoder_out_seq * K.expand_dims(inputs, -1), axis=1)
```

```
            if verbose:
                print('ci>', c_i.shape)
            return c_i, [c_i]

        def create_inital_state(inputs, hidden_size):
            # We are not using initial states, but need to pass something to
K.rnnfunciton
fake_state = K.zeros_like(inputs)  # <= (batch_size, enc_seq_len, latent_dim
fake_state = K.sum(fake_state, axis=[1, 2])  # <= (batch_size)
fake_state = K.expand_dims(fake_state)  # <= (batch_size, 1)
fake_state = K.tile(fake_state, [1, hidden_size])  # <= (batch_size, latent_dim
            return fake_state

fake_state_c = create_inital_state(encoder_out_seq, encoder_out_seq.shape[-1])
fake_state_e = create_inital_state(encoder_out_seq, encoder_out_seq.shape[1])  #
<= (batch_size, enc_seq_len, latent_dim

        """ Computing energy outputs """
        # e_outputs => (batch_size, de_seq_len, en_seq_len)
last_out, e_outputs, _ = K.rnn(
energy_step, decoder_out_seq, [fake_state_e],
        )

        """ Computing context vectors """
last_out, c_outputs, _ = K.rnn(
context_step, e_outputs, [fake_state_c],
        )

        return c_outputs, e_outputs

    def compute_output_shape(self, input_shape):
        """ Outputs produced by the layer """
        return [
tf.TensorShape((input_shape[1][0], input_shape[1][1], input_shape[1][2])),
tf.TensorShape((input_shape[1][0], input_shape[1][1], input_shape[0][1]))
        ]

decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(num_decoder_token, embedding_dim)
dec_emb = dec_emb_layer(decoder_inputs)
decoder_lstm = LSTM(units, return_sequences=True,
return_state=True,recurrent_initializer='glorot_uniform')
decoder_outputs, _, _ = decoder_lstm(dec_emb,
initial_state=encoder_states)
attn_layer = AttentionLayer()
```

```
attention_result,attention_weights =
attn_layer([encoder_outputs,decoder_outputs])

decoder_concat_input = Concatenate(axis=-
1,name='concat_layer')([decoder_outputs,attention_result])

decoder_dense = Dense(num_decoder_token, activation='softmax')
decoder_outputs = decoder_dense(decoder_concat_input)
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='rmsprop',
loss='categorical_crossentropy',metrics=['accuracy'])
model.summary()
model.fit([encoder_input_data, decoder_input_data], decoder_target_data,
batch_size=100, epochs=25)
encoder_model = Model(encoder_inputs, encoder_states)
encoder_model.summary()


decoder_state_input_h = Input(shape=(units,))
decoder_state_input_c = Input(shape=(units,))
decoder_states_inputs = [decoder_state_input_h,decoder_state_input_c]
decoder_hidden_state_input = Input(shape=(num_encoder_tokens,units))

dec_emb2= dec_emb_layer(decoder_inputs)
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2,
initial_state=decoder_states_inputs)

attention_result_inf, attention_weights_inf =
attn_layer([decoder_hidden_state_input,decoder_outputs2])
decoder_concatenate_input_inf = Concatenate(axis=-
1,name='concat_layer')([decoder_outputs2,attention_result_inf])

decoder_states2 = [state_h2, state_c2]
decoder_outputs2 = decoder_dense(decoder_concatenate_input_inf)

decoder_model = Model(
   [decoder_inputs]
+[decoder_hidden_state_input,decoder_state_input_h,decoder_state_input_c],
   [decoder_outputs2] + decoder_states2)
decoder_model.summary()
```

(iii) NMT with MBART transformers:

```
!pip install sacrebleu
!pip install git+https://github.com/huggingface/transformers -q
!pip install sentencepiece
import string
import re
import pandas as pd
from sacrebleu import sentence_bleu
from transformers import MBartForConditionalGeneration,
MBart50TokenizerFast
model = MBartForConditionalGeneration.from_pretrained("facebook/mbart-
large-50-one-to-many-mmt")
tokenizer = MBart50TokenizerFast.from_pretrained("facebook/mbart-large-50-
one-to-many-mmt", src_lang="en_XX")
df=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/Machine
Translation/hin.txt", sep='\t', header=None,
names=["english_sentence","hindi_sentence","path"])
df= df.drop(columns=['path'])
df.head()
def translate(sentence):
model_inputs = tokenizer(sentence, return_tensors="pt")
generated_tokens_ = model.generate(    **model_inputs,
forced_bos_token_id=tokenizer.lang_code_to_id["hi_IN"])
  trans = tokenizer.batch_decode(generated_tokens_, skip_special_tokens=True)
  return trans
df.isnull().sum()
df['english_sentence']=df['english_sentence'].apply(lambda x: x.lower())
df['hindi_sentence']=df['hindi_sentence'].apply(lambda x: x.lower())
df['english_sentence']=df['english_sentence'].apply(lambda x: x.strip())
df['hindi_sentence']=df['hindi_sentence'].apply(lambda x: x.strip())
df['english_sentence']=df['english_sentence'].apply(lambda x: re.sub(" +", " ", x))
df['hindi_sentence']=df['hindi_sentence'].apply(lambda x: re.sub(" +", " ", x))
df['english_sentence']=df['english_sentence'].apply(lambda x: ".join(ch for ch in x
if ch not in string.punctuation))
df['hindi_sentence']=df['hindi_sentence'].apply(lambda x: ".join(ch for ch in x if
ch not in string.punctuation))
df['english_sentence']=df['english_sentence'].str.replace('\d+', '')
df['hindi_sentence']=df['hindi_sentence'].str.replace('\d+', '')
df['predicted_hindi_sentence']=df['english_sentence'].apply(lambda x:
translate(x))
x = df.hindi_sentence.tolist()
k=2000
english =df.english_sentence[k]
actual = df.hindi_sentence[k]
predicted = df.predicted_hindi_sentence[k]
```

```
print("The actual english sentence is:",english)
print("The actual hindi sentence is:",actual)
print("The predicted hindi sentence is:",predicted)
print("The BLEU score is :",sentence_bleu(predicted,[actual]).score)
k=52
english =df.english_sentence[k]
actual = df.hindi_sentence[k]
predicted = df.predicted_hindi_sentence[k]
print("The actual english sentence is:",english)
print("The actual hindi sentence is:",actual)
print("The predicted hindi sentence is:",predicted)
print("The BLEU score is :",sentence_bleu(predicted,[actual]).score)
english = "birds eat"
correct='पंछीखाना'
hindi   = translate(english)
print("The english sentence is:",english)
print("The hindi sentence is:",hindi)
print("The BLEU score is :",sentence_bleu(hindi,[correct]).score)
```

Experiments:

(i) Environment and data:

Our project was implemented using Google Colab environment. This project uses TensorFlow as deep learning framework. We have used the English Hindi Dataset from Kaggle for the purpose of testing our model (link).

(ii) Evaluation:

We have used the Bi-Lingual Evaluation Understudy (BLEU) as well as the accuracy and loss as the parameter to check how well our model is working.

They were used to compare the three models as shown below:

Model	Accuracy	Loss	BLEU
Simple NMT	0.4384	0.7739	18.5
NMT with attention layer	0.4593	1.5094	-
Transformers	-	-	40.48

Fig: - These results were obtained against random samples for the purpose of proving. The accuracy and loss obtained from scikit-learn and BLEU score from SacreBLEU library.

Comparing the Results:

(i) NMT with LSTM :

```
test_sentence = 'birds eat'
encoder_test_data = np.zeros((len(df.english_sentence), max_input_length), dtype='float32')
for t, word in enumerate(test_sentence.split()):
    encoder_test_data[1,t] = input_token_index[word]
predicted=decode_sequence(encoder_test_data[1:2])
correct='<s> पंछी खाना। </s>'
print("The  english sentence is:",test_sentence)
print("The predicted hindi sentence is:",predicted)
print("The BLEU score is :",sentence_bleu(predicted,[correct]).score)
```

```
The  english sentence is: birds eat
The predicted hindi sentence is:  पंछी उड़ते हैं। </s>
The BLEU score is : 32.66828640925501
```

```
k=1301
english =df.english_sentence[k:k+1].values[0]
actual = df.hindi_sentence[k:k+1].values[0]
predicted = decode_sequence(encoder_input_data[k:k+1])
print("The actual english sentence is:",english)
print("The actual hindi sentence is:",actual)
print("The predicted hindi sentence is:",predicted)
print("The BLEU score is :",sentence_bleu(predicted,[actual]).score)
```

```
The actual english sentence is: you are not coming are you
The actual hindi sentence is: <s> तुम नहीं आ रहे हो ना </s>
The predicted hindi sentence is:  तुम तुम क्यों नहीं कर सकते। </s>
The BLEU score is : 21.53672420052281
```

```
k=2001
english =df.english_sentence[k]
actual = df.hindi_sentence[k]
predicted = df.predicted_hindi_sentence[k]
print("The actual english sentence is:",english)
print("The actual hindi sentence is:",actual)
print("The predicted hindi sentence is:",predicted)
print("The BLEU score is :",sentence_bleu(predicted,[actual]).score)
```

```
The actual english sentence is: the doctor advised him not to smoke
The actual hindi sentence is: डॉक्टर ने उसे सिगरेट न पीने की सलह दी।
The predicted hindi sentence is: ['डॉक्टर ने उसे सिखाया कि धूम्रपान न करें।']
The BLEU score is : 19.493995755254467
```

(ii) NMT with MBART model (using the same sentences as above) :

```
eng_text = "birds eat"
actual_hindi = "पक्षी खाते हैं"

model_inputs = tokenizer(eng_text, return_tensors="pt")

# translate from English to Hindi
generated_tokens = model.generate(
    **model_inputs,
    forced_bos_token_id=tokenizer.lang_code_to_id["hi_IN"]    #specifying the language
)

hindi_trans = tokenizer.batch_decode(generated_tokens, skip_special_tokens=True)
```

```
print("The actual english sentence is: ",eng_text)
print("The actual hindi sentence is: ",actual_hindi)
print("The predicted hindi sentence is:",hindi_trans)
print("The BLEU score is :",sentence_bleu(hindi_trans,[actual_hindi]).score)
```

```
The actual english sentence is:  birds eat
The actual hindi sentence is:  पक्षी खाते हैं
The predicted hindi sentence is: ['पक्षी खाना']
The BLEU score is : 30.326532985631665
```

```
eng_text = "you are not coming are you"
actual_hindi = "तुम नहीं आ रहे हो ना"

model_inputs = tokenizer(eng_text, return_tensors="pt")

# translate from English to Hindi
generated_tokens = model.generate(
    **model_inputs,
    forced_bos_token_id=tokenizer.lang_code_to_id["hi_IN"]    #specifying the language
)

hindi_trans = tokenizer.batch_decode(generated_tokens, skip_special_tokens=True)
```

```
print("The actual english sentence is: ",eng_text)
print("The actual hindi sentence is: ",actual_hindi)
print("The predicted hindi sentence is:",hindi_trans)
print("The BLEU score is :",sentence_bleu(hindi_trans,[actual_hindi]).score)
```

```
The actual english sentence is:  you are not coming are you
The actual hindi sentence is:  तुम नहीं आ रहे हो ना
The predicted hindi sentence is: ['तुम नहीं आ रहे हो क्या तुम']
The BLEU score is : 61.47881529512643
```

```
eng_text = "the doctor advised him not to smoke"
actual_hindi = "डॉक्टर ने उसे सिगरेट न पीने की सलह दी।"

model_inputs = tokenizer(eng_text, return_tensors="pt")

# translate from English to Hindi
generated_tokens = model.generate(
    **model_inputs,
    forced_bos_token_id=tokenizer.lang_code_to_id["hi_IN"]    #specifying the language
)

hindi_trans = tokenizer.batch_decode(generated_tokens, skip_special_tokens=True)
```

```
print("The actual english sentence is: ",eng_text)
print("The actual hindi sentence is: ",actual_hindi)
print("The predicted hindi sentence is:",hindi_trans)
print("The BLEU score is :",sentence_bleu(hindi_trans,[actual_hindi]).score)
```

```
The actual english sentence is:  the doctor advised him not to smoke
The actual hindi sentence is:  डॉक्टर ने उसे सिगरेट न पीने की सलह दी।
The predicted hindi sentence is: ['डॉक्टर ने उसे सिखाया कि धूम्रपान न करें।']
The BLEU score is : 19.493995755254467
```

Conclusion:

From the above project we understand that the NMT with attention layer and Transformers perform better than that of the simple NMT model. Since the transformer model we used in our project is pre-trained with almost all the sentence we assume that it has an accuracy of almost equal to 100 %. So, the transformer model performs well than these two models.

References:

1) Neural Machine Translation Using seq2seq model with Attention| by Aditya Shirsath | Medium | Geek Culture

2) NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE (DzmitryBahdanau Jacobs University Bremen, Germany KyungHyun Cho YoshuaBengio∗Universite de Montr ´ eal)

3) Research on Neural Machine Translation Model(Mengyao Chen, Yong Li, Runqi Li Institute of Information, Beijing University of Technology, No100 Pingleyuan, Chaoyang District, Beijing, China)

4) Hindi-English Neural Machine Translation Using Attention Model(Charu Verma, Aarti Singh, Swagata Seal, Varsha Singh, Iti Mathur)

5) ***Attention Is All You Need***
(Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, IlliaPolosukhin)
https://arxiv.org/pdf/1706.03762.pdf

6) https://jalammar.github.io/illustrated-transformer/

7) https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

8) ***Multilingual Denoising Pre-training for Neural Machine Translation***
(Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey EdunovMarjanGhazvininejad, Mike Lewis, Luke Zettlemoyer) Facebook AI Research
https://arxiv.org/abs/2001.08210

9) https://huggingface.co/transformers/model_doc/mbart.html