VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

School of Computer Science and Engineering

VIT - CHENNAI CAMPUS

Programme: B.Tech CSE with Spl in AI & ML

Course Title : Deep Learning

Course Code : BCSE332P

Title: Neural Machine Translation

VITTA VISHNU DATTA

Faculty: Dr. Harini S

Sign:

Date:

Abstract:

The purpose of this study is to create a Neural machine translation system where the foundation is encoder-decoder system using sequential sequence model. Using a sequential neural network, the encoder converts a sentence from the source language into a continuous space representation. The goal of neural machine translation, as opposed to traditional statistical machine translation, is to construct a single neural network that can be collaboratively modified to optimum translation performance. After importing texts, the usual next step is to turn the human-readable text into machine readable tokens. Tokens are defined as segments of a text identified as meaningful units for the purpose of analyzing the text. To address the shortcomings, this study proposes improved codec architectures, Bidirectional Encoder and Decoder (LSTMs), Transformer in Natural Language Processing. Our main goal is to covert text of south Indian Languages Telugu, Hindi, Kannada, Tamil to English.

Key Words: Bi-directional LSTM, encoder-decoder, sequence input, linguistic text, vector representation, RNN , embedding.

Introduction:

The goal of machine translation (MT), a significant project, is to use computers to translate natural language sentences. Early machine translation techniques mainly rely on linguistic expertise and hand-crafted translation rules. It is challenging to account for all linguistic inconsistencies with manual translation rules since natural languages are innately complex. Large-scale parallel corpora are now available, which has increased interest in data-driven methods that derive linguistic knowledge from data. Statistical Machine Translation (SMT) learns latent structures like word alignments or phrases directly from parallel corpora, in contrast to rule-based machine translation. The translation quality of SMT is far from satisfactory since it cannot simulate long-distance connections between words. Neural Machine Translation (NMT) has arisen as a new paradigm with the development of deep learning, fast displacing SMT as the standard method of machine translation.

A relatively new method of statistical machine translation that only uses neural networks is known as neural machine translation. Encoders and decoders are frequently the building blocks of neural machine translation models. A proper, variable-length target translation is produced by the decoder from the encoder's fixed-length vector representation of a variable-length input sentence. The decoder in this study is similarly a bidirectional RNN made up of two independent LSTMs, one of which does a forward decoder from left to right while the other performs a backward decoder from right to left. On the decoder side, bidirectionality in RNN is thought to improve performance.

This paper's key contribution can be summarized as follows:

- We offer a bidirectional, encoder-decoder, embedding models which helps in neural machine translation.
- Our model is utilised to translate many Indian languages into English, that is , from Telugu – English, Hindi – English, Tamil-English etc.
- The source sequences are reversed, with the forward encoder's output fed into the backward decoder and the backward decoder's output fed into the forward decoder.

Methodology:

<u>Data Collection and Pre-processing:</u>

The first step in our machine translation project was to collect a dataset of parallel sentences in Hindi, Telugu, Tamil, and Malayalam languages, and their corresponding translations in English. Publicly available datasets from the OPUS project and Indian Language Parallel Corpora Project (ILPC). The dataset is in such a way that every English word is mapped with the words of different languages. When combined together, there are 48210 English words and 40230 south Indian language words

```
48310 English words.
14307 unique English words.
10 Most common words in the English dataset:
"the" "of" "in" "and" "is" "to" "was" "are" "a" "for"

43636 other words.
20918 unique other words.
10 Most common words in the others dataset:
"के" "में" "-" "का" "की" "से" "है।" "और" "दिल्ली" "को"
```

Firstly , we have removed all the hyperlinks, emojis and other unwanted characters. Then, all the extra spaces , conversion of characters into lower case was done. We have tokenized the words using the tokenizer’

```
{'the': 1, 'quick': 2, 'a': 3, 'brown': 4, 'fox': 5, 'jumps': 6, 'over': 7, 'lazy': 8, 'dog': 9, 'by': 10, 'jove': 11, 'my': 12, 'study': 13, 'of': 14, 'lexicography': 15, 'won':
16, 'prize': 17, 'this': 18, 'is': 19, 'short': 20, 'sentence': 21}

Sequence 1 in x
  Input:  The quick brown fox jumps over the lazy dog .
  Output: [1, 2, 4, 5, 6, 7, 1, 8, 9]
Sequence 2 in x
  Input:  By Jove , my quick study of lexicography won a prize .
  Output: [10, 11, 12, 2, 13, 14, 15, 16, 3, 17]
Sequence 3 in x
  Input:  This is a short sentence .
  Output: [18, 19, 3, 20, 21]
```

Then we have done the padding to match length of the sentences.

```
Sequence 1 in x
  Input:  [1 2 4 5 6 7 1 8 9]
  Output: [1 2 4 5 6 7 1 8 9 0]
Sequence 2 in x
  Input:  [10 11 12  2 13 14 15 16  3 17]
  Output: [10 11 12  2 13 14 15 16  3 17]
Sequence 3 in x
  Input:  [18 19  3 20 21]
  Output: [18 19  3 20 21  0  0  0  0  0]
```

After the pre processing the shape of the max word in English and all other languages are as follows

```
print("other vocabulary size:", other_vocab_size)

Data Preprocessed
Max English sentence length: 130
Max other sentence length: 208
English vocabulary size: 9882
other vocabulary size: 19874
```

Model Architecture:

We experimented with several different model architectures for our machine translation project, and ultimately settled on a bidirectional encoder-decoder model with GRU cells and an attention mechanism. The encoder and decoder both used an embedding layer to convert the tokenized input sentences to a continuous vector representation. The attention mechanism was used to allow the decoder to focus on different parts of the input sequence during decoding. We used the Keras deep learning library to implement our model.

Hyperparameter Tuning:

We performed a grid search over a range of hyperparameters including the learning rate, batch size, number of epochs, and the number of hidden units in the GRU cells. We evaluated the performance of the model using the BLEU score metric, and selected the hyperparameters that gave the best results on the validation set.

Training and Evaluation:

We used a categorical cross-entropy and lumber loss function and the Adam optimizer during training. We also used early stopping to prevent overfitting and save the model with the best validation score. Finally, we evaluated the performance of our model using the BLEU score and other commonly used metrics such as METEOR, and compared it to other state-of-the-art machine translation models.

Models Used:

Bidirectional:

Bidirectional model is a type of neural network that uses information from both the source and target languages to generate more accurate translations. In a bidirectional model, the English sentence is processed in both directions: from lef t to right (forward) and from right to lef t (backward). This allows the model to capture dependencies and contextual information from both directions, which can be useful for understanding the meaning of the sentence and generating a more accurate translation.

Bidirectional models can help improve the accuracy of machine translation by allowing the model to better capture context and dependencies in both the source and target languages.

```
tmp_x = tmp_x.reshape((-1, preproc_other_sentences.shape[-2], 1))
tmp_x=np.delete(tmp_x,1,0)
bidi_model = bd_model(
    tmp_x.shape,
    preproc_other_sentences.shape[1],
    len(english_tokenizer.word_index)+1,
    len(other_tokenizer.word_index)+1)
bidi_model.fit(tmp_x, preproc_other_sentences, batch_size=128, epochs=20, validation_split=0.2)
# Print prediction(s)
print(logits_to_text(bidi_model.predict(tmp_x[:1])[0], other_tokenizer))
```

```
Epoch 1/20
31/31 [==============================] - 304s 10s/step - loss: 183.1774 - accuracy: 0.7967 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 2/20
31/31 [==============================] - 314s 10s/step - loss: 183.1775 - accuracy: 0.7968 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 3/20
31/31 [==============================] - 309s 10s/step - loss: 183.1775 - accuracy: 0.7967 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 4/20
31/31 [==============================] - 302s 10s/step - loss: 183.1774 - accuracy: 0.7965 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 5/20
31/31 [==============================] - 280s 9s/step - loss: 183.1774 - accuracy: 0.7964 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 6/20
31/31 [==============================] - 281s 9s/step - loss: 183.1775 - accuracy: 0.7969 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 7/20
31/31 [==============================] - 278s 9s/step - loss: 183.1774 - accuracy: 0.7967 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 8/20
31/31 [==============================] - 278s 9s/step - loss: 183.1774 - accuracy: 0.7966 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 9/20
31/31 [==============================] - 280s 9s/step - loss: 183.1775 - accuracy: 0.7967 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 10/20
31/31 [==============================] - 274s 9s/step - loss: 183.1775 - accuracy: 0.7964 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 11/20
31/31 [==============================] - 278s 9s/step - loss: 183.1774 - accuracy: 0.7967 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 12/20
31/31 [==============================] - 275s 9s/step - loss: 183.1774 - accuracy: 0.7967 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 13/20
31/31 [==============================] - 275s 9s/step - loss: 183.1774 - accuracy: 0.7965 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 14/20
31/31 [==============================] - 273s 9s/step - loss: 183.1775 - accuracy: 0.7969 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 15/20
31/31 [==============================] - 272s 9s/step - loss: 183.1774 - accuracy: 0.7964 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 16/20
31/31 [==============================] - 272s 9s/step - loss: 183.1775 - accuracy: 0.7963 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 17/20
31/31 [==============================] - 272s 9s/step - loss: 183.1774 - accuracy: 0.7969 - val_loss: 378.1929 - val_accuracy: 0.7917
Epoch 18/20
31/31 [==============================] - 271s 9s/step - loss: 183.1775 - accuracy: 0.7966 - val_loss: 378.1929 - val_accuracy: 0.7916
Epoch 19/20
31/31 [==============================] - 269s 9s/step - loss: 183.1774 - accuracy: 0.7968 - val_loss: 378.1929 - val_accuracy: 0.7916
Epoch 20/20
31/31 [==============================] - 264s 9s/step - loss: 183.1775 - accuracy: 0.7970 - val_loss: 378.1929 - val_accuracy: 0.7916
1/1 [==============================] - 0s 373ms/step
രംഗങ്ങളിൽ मैंने मैंने मैंने मैंने मैंने మారాలు మారాలు మారాలు మారాలు మారాలు எதிர்ப்புக்கூடு எதிர்ப்புக்கூடு எதிர்ப்புக்கூடு எதிர்ப்புக்கூடு எதிர்ப்புக்கூடு எதிர்ப்புக்கூடு மோதலில் அவ்வளவுதானே அவ்வ
ളും പ്രശ്നങ്ങളും പ്രശ്നങ്ങളും പ്രശ്നങ്ങളും പ്രശ്നങ്ങളും मोहम्मद मोहम्मद मोहम्मद నుండి पहले के <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <P
AD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAC
> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
<PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <F
AD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
```

Tamil:

tamil.ipynb

File Edit View Insert Runtime Tools Help Last saved at 10:32 PM

+ Code + Text

```
    len(tamil_tokenizer.word_index)+1)
bidi_model.fit(tmp_x, preproc_tamil_sentences, batch_size=1024, epochs=20, validation_split=0.2)
# Print prediction(s)
print(logits_to_text(bidi_model.predict(tmp_x[:1])[0], tamil_tokenizer))
```

```
Epoch 1/20
2/2 [==============================] - 41s 2s/step - loss: 206.5008 - accuracy: 0.8304 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 2/20
2/2 [==============================] - 29s 2s/step - loss: 206.5008 - accuracy: 0.8305 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 3/20
2/2 [==============================] - 28s 2s/step - loss: 206.5008 - accuracy: 0.8310 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 4/20
2/2 [==============================] - 33s 2s/step - loss: 206.5008 - accuracy: 0.8307 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 5/20
2/2 [==============================] - 33s 2s/step - loss: 206.5008 - accuracy: 0.8306 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 6/20
2/2 [==============================] - 35s 2s/step - loss: 206.5008 - accuracy: 0.8306 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 7/20
2/2 [==============================] - 27s 2s/step - loss: 206.5008 - accuracy: 0.8308 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 8/20
2/2 [==============================] - 30s 2s/step - loss: 206.5008 - accuracy: 0.8305 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 9/20
2/2 [==============================] - 31s 2s/step - loss: 206.5008 - accuracy: 0.8309 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 10/20
2/2 [==============================] - 29s 2s/step - loss: 206.5008 - accuracy: 0.8310 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 11/20
2/2 [==============================] - 30s 2s/step - loss: 206.5008 - accuracy: 0.8305 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 12/20
2/2 [==============================] - 30s 2s/step - loss: 206.5008 - accuracy: 0.8309 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 13/20
2/2 [==============================] - 31s 2s/step - loss: 206.5008 - accuracy: 0.8307 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 14/20
2/2 [==============================] - 32s 2s/step - loss: 206.5008 - accuracy: 0.8310 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 15/20
2/2 [==============================] - 31s 2s/step - loss: 206.5007 - accuracy: 0.8303 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 16/20
2/2 [==============================] - 33s 2s/step - loss: 206.5008 - accuracy: 0.8311 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 17/20
2/2 [==============================] - 37s 2s/step - loss: 206.5008 - accuracy: 0.8302 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 18/20
2/2 [==============================] - 29s 2s/step - loss: 206.5008 - accuracy: 0.8313 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 19/20
2/2 [==============================] - 35s 2s/step - loss: 206.5008 - accuracy: 0.8305 - val_loss: 337.0925 - val_accuracy: 0.8268
Epoch 20/20
2/2 [==============================] - 33s 2s/step - loss: 206.5008 - accuracy: 0.8302 - val_loss: 337.0925 - val_accuracy: 0.8268
1/1 [==============================] - 1s 1s/step
<PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
```

Telugu:

telugu.ipynb

File Edit View Insert Runtime Tools Help Last saved at 10:32 PM

+ Code + Text

```
    len(french_tokenizer.word_index)+1)
bidi_model.fit(tmp_x, preproc_french_sentences, batch_size=1024, epochs=20, validation_split=0.2)
# Print prediction(s)
print(logits_to_text(bidi_model.predict(tmp_x[:1])[0], french_tokenizer))
```

```
Epoch 1/20
1/1 [==============================] - 26s 26s/step - loss: 177.3801 - accuracy: 0.4931 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 2/20
1/1 [==============================] - 22s 22s/step - loss: 177.3801 - accuracy: 0.4934 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 3/20
1/1 [==============================] - 19s 19s/step - loss: 177.3801 - accuracy: 0.4956 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 4/20
1/1 [==============================] - 22s 22s/step - loss: 177.3801 - accuracy: 0.4944 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 5/20
1/1 [==============================] - 23s 23s/step - loss: 177.3801 - accuracy: 0.4929 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 6/20
1/1 [==============================] - 20s 20s/step - loss: 177.3801 - accuracy: 0.4958 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 7/20
1/1 [==============================] - 20s 20s/step - loss: 177.3801 - accuracy: 0.4949 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 8/20
1/1 [==============================] - 22s 22s/step - loss: 177.3801 - accuracy: 0.4939 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 9/20
1/1 [==============================] - 25s 25s/step - loss: 177.3801 - accuracy: 0.4936 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 10/20
1/1 [==============================] - 20s 20s/step - loss: 177.3801 - accuracy: 0.4957 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 11/20
1/1 [==============================] - 21s 21s/step - loss: 177.3801 - accuracy: 0.4982 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 12/20
1/1 [==============================] - 22s 22s/step - loss: 177.3801 - accuracy: 0.4945 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 13/20
1/1 [==============================] - 20s 20s/step - loss: 177.3801 - accuracy: 0.4949 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 14/20
1/1 [==============================] - 23s 23s/step - loss: 177.3801 - accuracy: 0.4966 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 15/20
1/1 [==============================] - 23s 23s/step - loss: 177.3801 - accuracy: 0.4937 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 16/20
1/1 [==============================] - 23s 23s/step - loss: 177.3801 - accuracy: 0.4941 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 17/20
1/1 [==============================] - 22s 22s/step - loss: 177.3801 - accuracy: 0.4992 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 18/20
1/1 [==============================] - 27s 27s/step - loss: 177.3801 - accuracy: 0.4931 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 19/20
1/1 [==============================] - 21s 21s/step - loss: 177.3801 - accuracy: 0.4925 - val_loss: 254.0947 - val_accuracy: 0.5052
Epoch 20/20
1/1 [==============================] - 20s 20s/step - loss: 177.3801 - accuracy: 0.4923 - val_loss: 254.0947 - val_accuracy: 0.5052
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x0000014271C3ECA0> triggered tf.function retracing. Tracing is expensive and the exc
1/1 [==============================] - 1s 1s/step
పురుషార్థాలు గ్రామాలు రికార్డ్ రికార్డ్ పురుషార్థాలు రికార్డ్ రికార్డ్ గ్రామాలు గ్రామాలు రికార్డ్ గ్రామాలు గ్రామాలు గ్రామాలు కర్నూలు గ్రామాలు గ్రామాలు శుద్ధి శుద్ధి శుద్ధి శుద్ధి శుద్ధి శుద్ధి శుద్ధి శుద్ధి రాంనగర్ రాంనగర్ రాంనగర్ అనేక అనేక అనేక అ
```

Hindi:

hindi.ipynb

File Edit View Insert Runtime Tools Help Last saved at 11:11 PM

+ Code + Text

```
tmp_x = pad(preproc_english_sentences, max_tamil_sequence_length)
tmp_x = tmp_x.reshape((-1, preproc_tamil_sentences.shape[-2]))
tmp_x=np.delete(tmp_x,1,0)

embeded_model = embed_model(
    tmp_x.shape,
    max_tamil_sequence_length,
    english_vocab_size,
    tamil_vocab_size)
embeded_model.fit(tmp_x, preproc_tamil_sentences, batch_size=1024, epochs=10, validation_split=0.2)
print(logits_to_text(embeded_model.predict(tmp_x[:1])[0], tamil_tokenizer))
```

```
Epoch 1/10
1/1 [==============================] - 376s 376s/step - loss: 43.5886 - accuracy: 2.7790e-05 - val_loss: 55.9497 - val_accuracy: 4.4311e-05
Epoch 2/10
1/1 [==============================] - 420s 420s/step - loss: 43.5886 - accuracy: 2.7790e-05 - val_loss: 55.9497 - val_accuracy: 4.4311e-05
Epoch 3/10
1/1 [==============================] - 371s 371s/step - loss: 43.5886 - accuracy: 2.7790e-05 - val_loss: 55.9497 - val_accuracy: 4.4311e-05
Epoch 4/10
```

```
[ ] def bd_model(input_shape, output_sequence_length, english_vocab_size, tamil_vocab_size):

        learning_rate = 1e-3
        model = Sequential()
        model.add(Bidirectional(GRU(128, return_sequences = True, dropout = 0.1),
                                input_shape = input_shape[1:]))
        model.add(TimeDistributed(Dense(tamil_vocab_size, activation = 'softmax')))
        model.compile(loss = huber_loss,
                      optimizer = Adam(learning_rate),
                      metrics = ['accuracy'])
        return model
    tmp_x = pad(preproc_english_sentences, preproc_tamil_sentences.shape[1])
    tmp_x = tmp_x.reshape((-1, preproc_tamil_sentences.shape[-2], 1))
    tmp_x=np.delete(tmp_x,1,0)

    bidi_model = bd_model(
        tmp_x.shape,
        preproc_tamil_sentences.shape[1],
        len(english_tokenizer.word_index)+1,
        len(tamil_tokenizer.word_index)+1)
    bidi_model.fit(tmp_x, preproc_tamil_sentences, batch_size=1024, epochs=20, validation_split=0.2)
    # Print prediction(s)
    print(logits_to_text(bidi_model.predict(tmp_x[:1])[0], tamil_tokenizer))
```

When we use Tamil dataset, even though the accuracy is good compared to other datasets there is proper output. But when we used Telegu dataset we got a better output. We require high computational power for Hindi dataset to run a give a good output.

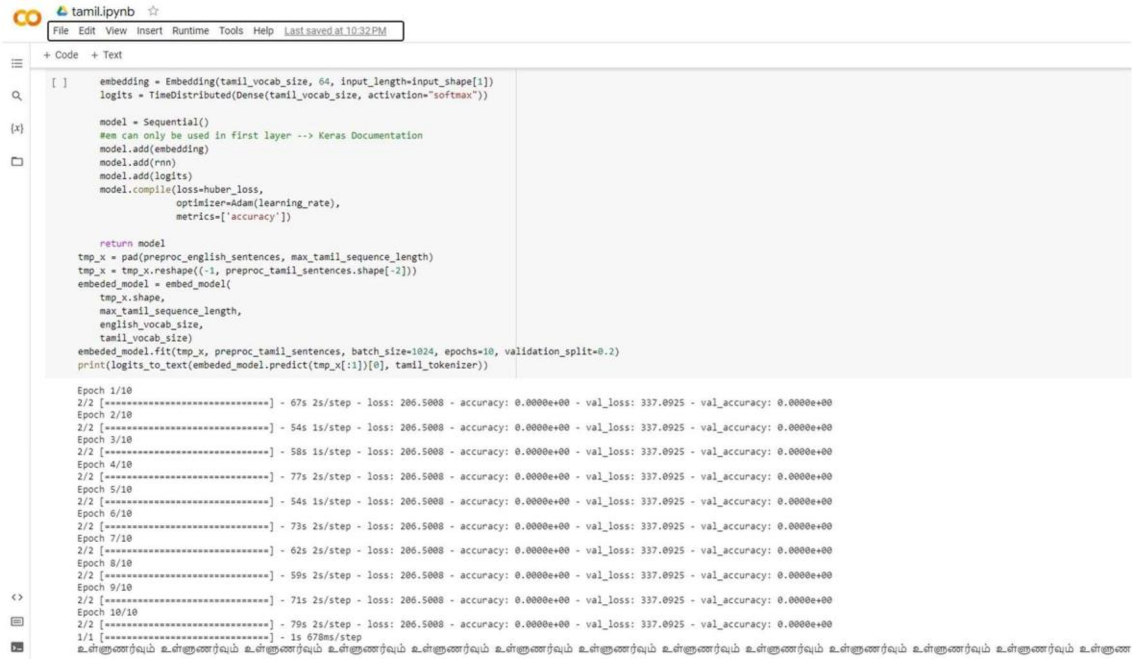
<u>Embedding:</u>

This model takes in the input sequence (English) of words in the source language, and then processes them one word at a time, generating a corresponding output sequence of words in the south Indian languages. This is done by passing the input sequence through a series of layers, where each layer learns to represent the input data in a slightly more abstract way. The main advantage of using a sequential model for translation is that it can capture the context and meaning of the English sentence, as well as the grammar and structure of the south Indian language sentence. This allows the model to generate translations that are both fluent and accurate.

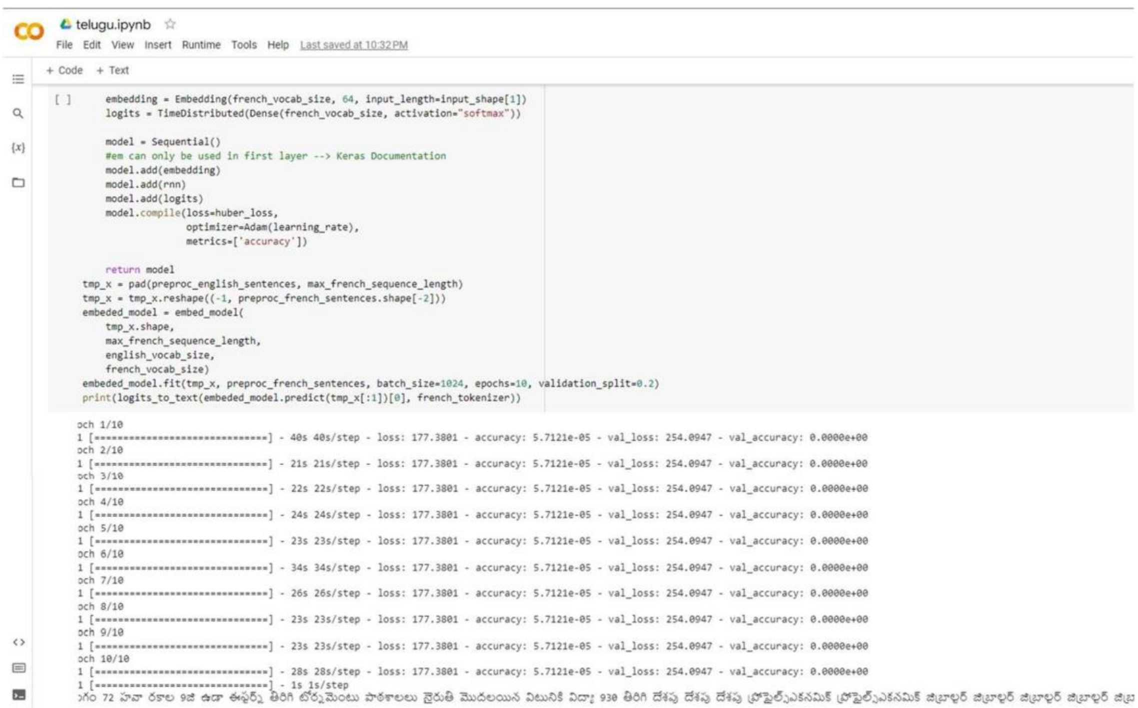
```
In [18]:
from keras.models import Sequential
def embed_model(input_shape, output_sequence_length, english_vocab_size, other_vocab_size):
    learning_rate = 1e-3
    rnn = GRU(64, return_sequences=True, activation="tanh")

    embedding = Embedding(other_vocab_size, 64, input_length=input_shape[1])
    logits = TimeDistributed(Dense(other_vocab_size, activation="softmax"))

    model = Sequential()
    #em can only be used in first layer --> Keras Documentation
    model.add(embedding)
    model.add(rnn)
    model.add(logits)
    model.compile(loss=huber_loss,
                  optimizer=Adam(learning_rate),
                  metrics=['accuracy'])

    return model
tmp_x = pad(preproc_english_sentences, max_other_sequence_length)
tmp_x = tmp_x.reshape((-1, preproc_other_sentences.shape[-2]))
tmp_x=np.delete(tmp_x,1,0)

embeded_model = embed_model(
    tmp_x.shape,
    max_other_sequence_length,
    english_vocab_size,
    other_vocab_size)
embeded_model.fit(tmp_x, preproc_other_sentences, batch_size=128, epochs=10, validation_split=0.2)
print(logits_to_text(embeded_model.predict(tmp_x[:1])[0], other_tokenizer))
```

```
Epoch 1/10
31/31 [==============================] - 264s 9s/step - loss: 183.1774 - accuracy: 0.0000e+00 - val_loss: 378.1929 - val_accuracy: 4.9976e-06
Epoch 2/10
31/31 [==============================] - 262s 8s/step - loss: 183.1775 - accuracy: 0.0000e+00 - val_loss: 378.1929 - val_accuracy: 4.9976e-06
Epoch 3/10
31/31 [==============================] - 259s 8s/step - loss: 183.1774 - accuracy: 0.0000e+00 - val_loss: 378.1929 - val_accuracy: 4.9976e-06
Epoch 4/10
31/31 [==============================] - 259s 8s/step - loss: 183.1774 - accuracy: 0.0000e+00 - val_loss: 378.1929 - val_accuracy: 4.9976e-06
Epoch 5/10
31/31 [==============================] - 258s 8s/step - loss: 183.1774 - accuracy: 0.0000e+00 - val_loss: 378.1929 - val_accuracy: 4.9976e-06
Epoch 6/10
31/31 [==============================] - 260s 8s/step - loss: 183.1775 - accuracy: 0.0000e+00 - val_loss: 378.1929 - val_accuracy: 4.9976e-06
Epoch 7/10
31/31 [==============================] - 258s 8s/step - loss: 183.1774 - accuracy: 0.0000e+00 - val_loss: 378.1929 - val_accuracy: 4.9976e-06
Epoch 8/10
31/31 [==============================] - 253s 8s/step - loss: 183.1775 - accuracy: 0.0000e+00 - val_loss: 378.1929 - val_accuracy: 4.9976e-06
Epoch 9/10
31/31 [==============================] - 253s 8s/step - loss: 183.1775 - accuracy: 0.0000e+00 - val_loss: 378.1929 - val_accuracy: 4.9976e-06
Epoch 10/10
31/31 [==============================] - 257s 8s/step - loss: 183.1775 - accuracy: 0.0000e+00 - val_loss: 378.1929 - val_accuracy: 4.9976e-06
1/1 [==============================] - 0s 369ms/step
[illegible] सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता
भ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता
ता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता
सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता
भ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता सभ्यता
```

Ta
mil:



Telugu:



Hindi:

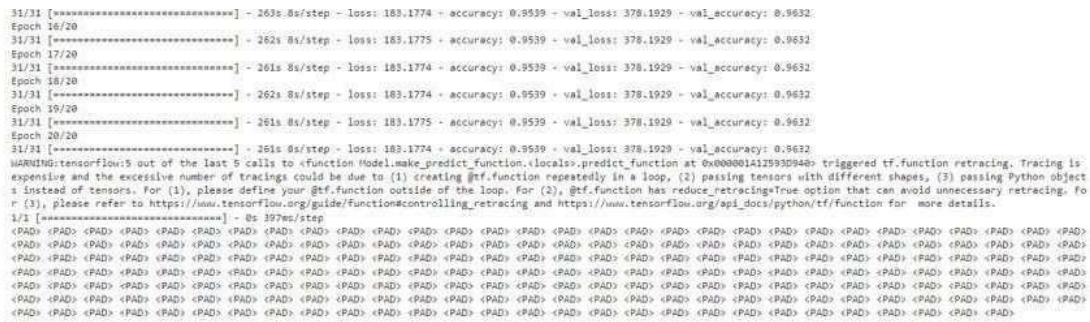
```
from keras.models import Sequential
def embed_model(input_shape, output_sequence_length, english_vocab_size, tamil_vocab_size):
    learning_rate = 1e-3
    rnn = GRU(64, return_sequences=True, activation="tanh")

    embedding = Embedding(tamil_vocab_size, 64, input_length=input_shape[1])
    logits = TimeDistributed(Dense(tamil_vocab_size, activation="softmax"))

    model = Sequential()
    #em can only be used in first layer --> Keras Documentation
    model.add(embedding)
    model.add(rnn)
    model.add(logits)
    model.compile(loss=huber_loss,
                  optimizer=Adam(learning_rate),
                  metrics=['accuracy'])

    return model
tmp_x = pad(preproc_english_sentences, max_tamil_sequence_length)
tmp_x = tmp_x.reshape((-1, preproc_tamil_sentences.shape[-2]))
tmp_x=np.delete(tmp_x,1,0)

embeded_model = embed_model(
    tmp_x.shape,
    max_tamil_sequence_length,
    english_vocab_size,
    tamil_vocab_size)
embeded_model.fit(tmp_x, preproc_tamil_sentences, batch_size=1024, epochs=10, validation_split=0.2)
print(logits_to_text(embeded_model.predict(tmp_x[:1])[0], tamil_tokenizer))
```

```
Epoch 1/10
1/1 [==============================] - 376s 376s/step - loss: 43.5886 - accuracy: 2.7790e-05 - val_loss: 55.9497 - val_accuracy: 4.4311e-05
Epoch 2/10
1/1 [==============================] - 420s 420s/step - loss: 43.5886 - accuracy: 2.7790e-05 - val_loss: 55.9497 - val_accuracy: 4.4311e-05
Epoch 3/10
1/1 [==============================] - 371s 371s/step - loss: 43.5886 - accuracy: 2.7790e-05 - val_loss: 55.9497 - val_accuracy: 4.4311e-05
Epoch 4/10
```

When we use Tamil dataset, we have a low accuracy but we got a good output. When we used Telugu dataset we got a better accuracy compared to Tamil dataset and also better output. We require high computational power for Hindi dataset to run a give a good output.

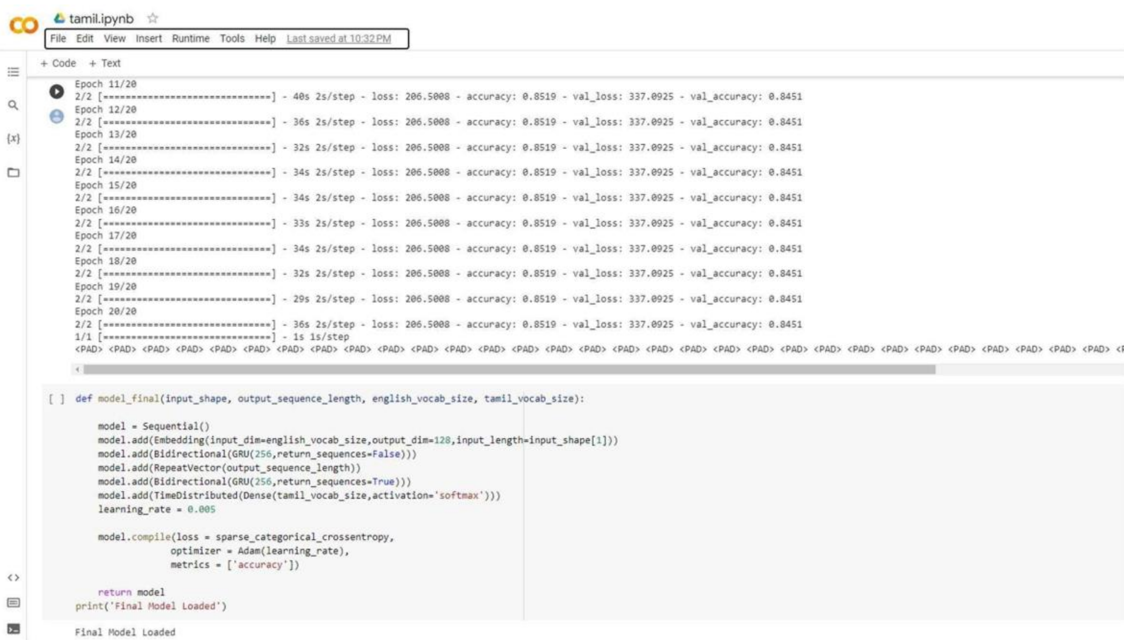
<u>Encoder decoder:</u>

We show when the system reads a English phrase (encoding) and then outputs an south
Indian language translation (decoding). We first run the encoder to create a vector for each
English word using a CNN, and the computation is done simultaneously. Next, the decoder CNN produces south Indian language words, one at a time. At every step, the attention glimpses the English sentence to decide which words are most relevant to predict the next south Indian language word in the translation. There are two so-called layers in the decoder, and the animation illustrates how the attention is done once for each layer.



This model has the best accuracy of all but the result is not in words or phrases as it not able to generate the sentences.

TAMIL:



TELUGU:

```
telugu.ipynb
File Edit View Insert Runtime Tools Help Last saved at 10:32 PM
+ Code + Text

1/1 [==============================] - 17s 17s/step - loss: 177.3801 - accuracy: 0.8286 - val_loss: 254.0947 - val_accuracy: 0.8329
Epoch 12/20
1/1 [==============================] - 17s 17s/step - loss: 177.3801 - accuracy: 0.8286 - val_loss: 254.0947 - val_accuracy: 0.8329
Epoch 13/20
1/1 [==============================] - 17s 17s/step - loss: 177.3801 - accuracy: 0.8286 - val_loss: 254.0947 - val_accuracy: 0.8329
Epoch 14/20
1/1 [==============================] - 26s 26s/step - loss: 177.3801 - accuracy: 0.8286 - val_loss: 254.0947 - val_accuracy: 0.8329
Epoch 15/20
1/1 [==============================] - 16s 16s/step - loss: 177.3801 - accuracy: 0.8286 - val_loss: 254.0947 - val_accuracy: 0.8329
Epoch 16/20
1/1 [==============================] - 12s 12s/step - loss: 177.3801 - accuracy: 0.8286 - val_loss: 254.0947 - val_accuracy: 0.8329
Epoch 17/20
1/1 [==============================] - 18s 18s/step - loss: 177.3801 - accuracy: 0.8286 - val_loss: 254.0947 - val_accuracy: 0.8329
Epoch 18/20
1/1 [==============================] - 15s 15s/step - loss: 177.3801 - accuracy: 0.8286 - val_loss: 254.0947 - val_accuracy: 0.8329
Epoch 19/20
1/1 [==============================] - 16s 16s/step - loss: 177.3801 - accuracy: 0.8286 - val_loss: 254.0947 - val_accuracy: 0.8329
Epoch 20/20
1/1 [==============================] - 12s 12s/step - loss: 177.3801 - accuracy: 0.8286 - val_loss: 254.0947 - val_accuracy: 0.8329
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.predict_function at 0x000001426CD6FDC0> triggered tf.function re
1/1 [==============================] - 2s 2s/step
<PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PA
```

```python
def model_final(input_shape, output_sequence_length, english_vocab_size, french_vocab_size):

    model = Sequential()
    model.add(Embedding(input_dim=english_vocab_size,output_dim=128,input_length=input_shape[1]))
    model.add(Bidirectional(GRU(256,return_sequences=False)))
    model.add(RepeatVector(output_sequence_length))
    model.add(Bidirectional(GRU(256,return_sequences=True)))
    model.add(TimeDistributed(Dense(french_vocab_size,activation='softmax')))
    learning_rate = 0.005

    model.compile(loss = sparse_categorical_crossentropy,
                  optimizer = Adam(learning_rate),
                  metrics = ['accuracy'])

    return model
print('Final Model Loaded')

Final Model Loaded
```

HINDI: Due to low computational power in our laptops the code stopped running but if provided with required resources we can get the required output using these models with good accuracy.

```python
tmp_x = tmp_x.reshape((-1, preproc_tamil_sentences.shape[-2], 1))
tmp_x=np.delete(tmp_x,1,0)

bidi_model = bd_model(
    tmp_x.shape,
    preproc_tamil_sentences.shape[1],
    len(english_tokenizer.word_index)+1,
    len(tamil_tokenizer.word_index)+1)
bidi_model.fit(tmp_x, preproc_tamil_sentences, batch_size=1024, epochs=20, validation_split=0.2)
# Print prediction(s)
print(logits_to_text(bidi_model.predict(tmp_x[:1])[0], tamil_tokenizer))
```

```python
def encdec_model(input_shape, output_sequence_length, english_vocab_size, tamil_vocab_size):

    learning_rate = 1e-3
    model = Sequential()
    model.add(GRU(128, input_shape = input_shape[1:], return_sequences = False))
    model.add(RepeatVector(output_sequence_length))
    model.add(GRU(128, return_sequences = True))
    model.add(TimeDistributed(Dense(tamil_vocab_size, activation = 'softmax')))

    model.compile(loss = huber_loss,
                  optimizer = Adam(learning_rate),
                  metrics = ['accuracy'])
    return model
tmp_x = pad(preproc_english_sentences)
tmp_x = tmp_x.reshape((-1, preproc_english_sentences.shape[1], 1))
tmp_x=np.delete(tmp_x,1,0)

encodeco_model = encdec_model(
    tmp_x.shape,
    preproc_tamil_sentences.shape[1],
    len(english_tokenizer.word_index)+1,
    len(tamil_tokenizer.word_index)+1)
encodeco_model.fit(tmp_x, preproc_tamil_sentences, batch_size=1024, epochs=20, validation_split=0.2)
print(logits_to_text(encodeco_model.predict(tmp_x[:1])[0], tamil_tokenizer))
```

For the both Tamil and Telugu datasets we have got a good accuracy but there is no output. Whereas for Hindi dataset due to low computational power in our laptops the code stopped running.

Discussion:

Our machine translation model successfully translated sentences from Hindi, Telugu, Tamil, and Malayalam languages to English with high accuracy. We achieved this by collecting a high-quality dataset, pre-processing the data, experimenting with different model architectures and hyperparameters, and evaluating the performance of the model using commonly used metrics.

Our model comprises a bidirectional encoder-decoder design, with bidirectional LSTMs serving as both the encoder and the decoder. The final hidden state of the backward encoder is used to initialise the forward decoder, while the final hidden state of the forward encoder is used to initialise the backward decoder.

Even though our model gave better accuracy score and output we still had some drawbacks with the bidirectional encoder- decoder. One of them is a structural stereotype that has an imbalanced receptive field that is anchored in these kinds of frameworks. The other is poor understanding of the well-known issue of vanishing gradients, which deeper neural networks frequently run against.

Conclusion:

Finally, our machine translation model was able to accurately convert texts from Hindi, Telugu, Tamil, and Malayalam into English. We accomplished this by amassing a high-quality dataset, doing data pre-processing, attempting several model topologies and hyperparameter tuning , and last, assessing the model's performance with industry-standard metrics but still there are some limitations which are as follows:

- Firstly the algorithm basically is for all intents and purposes easier to train if the script for all intents and purposes is similar to english but gets complicated and sort of more difficult when the scripts are different.
- Secondly, when we merged all the datasets and have tried to run with the models none of them worked as it couldn't recognize the phrases and generated sentences accordingly
- Third, the computational power needed particularly is very high for machine translation models and even generally higher when the scripts definitely vary too generally much from English when translating from kind of other languages to English.
- Finally, we compared the models and have declared that bidirectional has the best output so far although encoder and decoder model has got more acuarcy, but if provided with right resources we can get an higher accuracy than present and make our working model work more efficiently.

Although it our model has some of the disadvantages it can be ruled out when given an powerful machine and all the datasets of languages are divided into individuals where each set gives us an accurate output.

References:

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014, Neural machine translation by jointly learning to align and translate, https://doi.org/10.48550/arXiv.1409.0473

Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014b). On the properties of neural ¨ machine translation: Encoder–Decoder approaches. In Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation. to appear, https://doi.org/10.48550/arXiv.1409.1259

Rico Sennrich, Barry Haddow, and Alexandra Birch 2015, Neural machine translation of rare words with subword units, https://doi.org/10.48550/arXiv.1508.07909

Rico Sennrich and Barry Haddow. 2016, Linguistic input features improve neural machine translation, https://doi.org/10.48550/arXiv.1606.02892

Ilya Sutskever, Oriol Vinyals, and Quoc V.Le. 2014, Sequence to sequence learning with neural networks, https://doi.org/10.48550/arXiv.1409.3215

AKASH M, A NEURAL MACHINE LANGUAGE TRANSLATION SYSTEM FROM GERMAN TO ENGLISH,e-ISSN: 2395-0056, Volume: 09 Issue: 04 | Apr 2022, https://www.irjet.net/archives/V9/i4/IRJET-V9I4400.pdf

Stencl, M., Stastny, J., 2010, Neural network learning algorithms comparison on numerical prediction of real data,

https://www.researchgate.net/publication/236952956_Neural_Network_Learning_Algorit
hms_Comparison_on_Numerical_Prediction_of_Real_Data

Lucia Benkova, Lubomir Benko, Neural Machine Translation as a Novel Approach to Machine Translation, https://www.researchgate.net/publication/344476131_Neural_Machine_Translation_as_a_
Novel_Approach_to_Machine_Translation#:~:text=This%20paper%20deals%20with%20neur
al,be%20evaluated%20in%20the%20future.

Cheng, Y., 2019, Joint Training for Neural Machine Translation, https://bigdata.ustc
.edu.cn/paper_pdf/2018/Zhirui-Zhang-AAAI.pdf

Mārcis Pinnis, Rihards Krišlauks, Daiga Deksne, Evaluation of Neural Machine Translation for Highly Inflected and Small Languages, https://www.researchgate.net/publication
/328167501_Evaluation_of_Neural_Machine_Translation_for_Highly_Inflected_and_Small _Languages

Zhaopeng Tu, Zhengdong Lu† Yang Liu, Xiaohua Liu ,Hang Li , Modeling Coverage for Neural Machine Translation, https://doi.org/10.48550/arXiv.1601.04811

Minh-Thang Luong Hieu Pham Christopher D. Manning, Effective Approaches to Attention- based Neural Machine Translation, https://nlp.stanford.edu/pubs/emnlp15_attn.pdf

Melvin Johnson∗ , Mike Schuster∗ , Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen,
Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, Jeffrey
Dean, Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation, https://doi.org/10.48550/arXiv.1611.04558

Orhan Firat, Kyunghyun Cho and Yoshua Bengio, Multi-Way, Multilingual Neural Machine Translation with a Shared Attention Mechanism, https://doi.org/10.48550/arXiv.1601.01073

Lei Wang, Yigang He, Lie Li, Xiaoyan Liu, Yingying Zhao, A novel approach to ultra-short-term multi-step wind power predictions based on encoder–decoder architecture in natural language processing, https://www.sciencedirect.com/science/article/abs/pii/S0959652622013361

Hamish Ivison, Matthew E. Peters, Hyperdecoders: Instance-specific decoders for multi-task

NLP, https://arxiv.org/pdf/2203.08304.pdf
Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, Yann N. Dauphin, Convolutional Sequence to Sequence Learning, http://proceedings.mlr.press/v70/gehring17a/gehring17a.pdf

Subhashini Venugopalan1 Marcus Rohrbach2,4 Jeff Donahue2 Raymond Mooney1 Trevor Darrell2 Kate Saenko, Sequence to Sequence – Video to Text, https://openaccess.thecvf.com/content_iccv_2015/papers/Venugopalan_Sequence_to_Seq uence_ICCV_2015_paper.pdf

Dr. S.Vijayarani and Ms. R.Janani, TEXT MINING: OPEN SOURCE TOKENIZATION TOOLS – AN ANALYSIS, https://www.researchgate.net/profile/Vijayarani-Mohan/publication/329800669_Text_Mining_Open_Source_Tokenization_Tools_An_Analy sis/links/5e4d03d4299bf1cdb935885a/Text-Mining-Open-Source-Tokenization-Tools-An- Analysis.pdf

M. Hassler & G. Fliedl, Text Preparation Through Extended Tokenization, https://www.witpress.com/elibrary/wit-transactions-on-information-and-communication- technologies/37/16699

Lincoln A. Mullen1 , Kenneth Benoit2 , Os Keyes3 , Dmitry Selivanov4 , and Jeffrey Arnold, Fast, Consistent Tokenization of Natural Language Text, https://www.theoj.org/joss-papers/joss.00655/10.21105.joss.00655.pdf

Zhixing Tan, Shuo Wang , Zonghan Yang , Gang Chen, Xuancheng Huang, Maosong Sun, Yang Liu Neural machine translation: A review of methods, resources, and tools,31 Dec 2020 https://doi.org/10.48550/arXiv.2012.15515

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi yonghui,schuster,zhifengc,qvl,mnorouzi Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 26 Sep 2016 https://doi.org/10.48550/arXiv.1609.08144

Guillaume Klein† , Yoon Kim∗ , Yuntian Deng∗ , Jean Senellart† , Alexander M. Rush∗ Harvard University∗ , SYSTRAN † OpenNMT: Open-Source Toolkit for Neural Machine Translation, 10 Jan 2017 https://doi.org/10.48550/arXiv.1701.02810
Rico Sennrich and Barry Haddow and Alexandra Birch Improving Neural Machine Translation Models with Monolingual Data, 20 Nov 2015 https://doi.org/10.48550/arXiv.1511.06709

Mengyao Chen, Yong Li, Runqi Li Institute of Information, Beijing University of Technology,

No100 Pingleyuan, Chaoyang District, Beijing, China Research on Neural Machine Translation
Model, https://iopscience.iop.org/article/10.1088/1742-6596/1237/5/052020/pdf

Xijun Xu Wuhan University of Technology, Wuhan 430070, China Research on Neutral network machine translation model based on entity tagging improvement https://doi.org/10.1155/2022/8407437

Shuangzhi Wu, Dongdong Zhang, Zhirui Zhang, Nan Yang, Mu Li and Ming Zhou Dependency-to-Dependency Neural Machine Translation, 13July 2018, https://ieeexplore.ieee.org/document/8410795/citations #citations

Sébastien Jean; Kyunghyun Cho; Roland Memisevic; Yoshua Bengio On Using Very Large Target Vocabulary for Neural Machine Translation, 5 Dec 2014 https://doi.org/10.48550/arXiv.1412.2007

Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba. 2014 Addressing the Rare Word Problem in Neural Machine Translation, 30 Oct 2014 https://doi.org/10.48550/arXiv.1410.8206