Machine translation using natural language processing

Middi Venkata Sai Rishita[1,*], *Middi Appala* Raju[2], and *Tanvir Ahmed* Harris[3]

[1]RNS Institute of Technology, India
[2]Christ University, India
[3]IIT Bombay, India

Abstract. Machine Translation is the translation of text or speech by a computer with no human involvement. It is a popular topic in research with different methods being created, like rule-based, statistical and example-based machine translation. Neural networks have made a leap forward to machine translation. This paper discusses the building of a deep neural network that functions as a part of end-to-end translation pipeline. The completed pipeline would accept English text as input and return the French Translation. The project has three main parts which are preprocessing, creation of models and Running the model on English Text.

1 Introduction

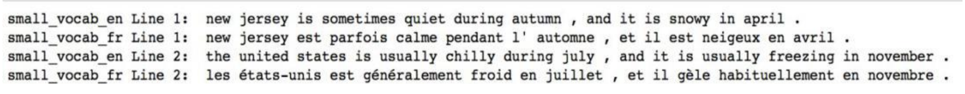
Machine translation (MT) is a domain of computational linguistics, which explores the use of software to translate text or speech from language to another. Machine translation simply performs substitution of words in one language for words in other, but that may not assure good translation. A more sophisticated method which is also a growing field used to address the issue of recognition of multiple phrases is with statistical and neural technique. In this translation of text from one language to another, there is no human involvement and it is the machine which performs the process of conversion. There are three types of machine translation system-rules based, statistical and neural. Rule based is a conventional method which is a combination of language and grammar and the support of dictionaries. This work focusses on building an end to end machine translation pipeline. We have discussed multiple existing architectures and finally proposed a hybrid model to achieve a more powerful system for machine translation from English to French.

2 Part 1: Dataset

The first step in the implementation of any Deep Learning project is the investigation of DataSet that would be used to train the pipeline as well as evaluate the pipeline. Most commonly, the datasets used for Machine Translation are from WMT (the website that is dedicated to research in statistical machine translation). Due to time constraints, the dataset contains small vocabulary. This facilitates the training in a reasonable time.

[*] Corresponding author: rishimiddi@gmail.com

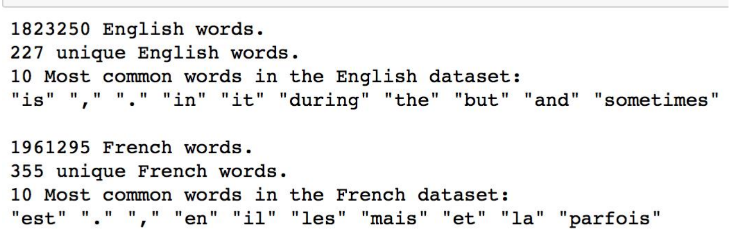
The data located in a filepath is loaded. The file contains English sentences with their French translations. Load the data from these files. Print the first two lines from each file.



```
small_vocab_en Line 1:  new jersey is sometimes quiet during autumn , and it is snowy in april .
small_vocab_fr Line 1:  new jersey est parfois calme pendant l' automne , et il est neigeux en avril .
small_vocab_en Line 2:  the united states is usually chilly during july , and it is usually freezing in november .
small_vocab_fr Line 2:  les états-unis est généralement froid en juillet , et il gèle habituellement en novembre .
```

Fig. 1. First two lines of English and French from each file.

In the figure above, the punctuations have been delimited using spaces. The conversion to lowercase has been done. The complexity of vocabulary determines the complexity of the problem. The complexity of the dataset is given below.



```
1823250 English words.
227 unique English words.
10 Most common words in the English dataset:
"is" "," "." "in" "it" "during" "the" "but" "and" "sometimes"

1961295 French words.
355 unique French words.
10 Most common words in the French dataset:
"est" "." "," "en" "il" "les" "mais" "et" "la" "parfois"
```

Fig. 2. The complexity of dataset.

3 Part 2: Pre-process the data

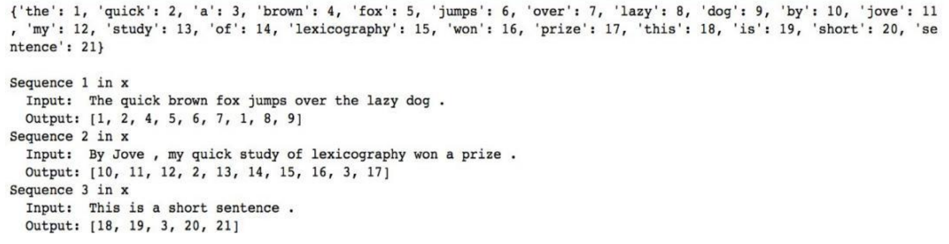
The text data is not used as input to the model. The text is converted into sequences of integers using the pre-process methods which are Tokenization and addition of Padding.

3.1 Tokenize (Implementation)

The neural network has to comprehend the input data for it to predict on text data. The network can understand ASCII characters. The text data like “dog” is a sequence of ASCII character encodings. The operations performed on network include multiplication and addition operations, hence, text is converted to numbers.

Each character and word is assigned a number, conventionally called character IDs and word IDs, respectively. A word level model used word ids that generate text predictions for every word. This has lower complexity compared to character level model.

Keras has a utility class known as Tokenizer. This class allows to vectorize a text corpus, by turning each text into either a sequence of integers or into a vector where the coefficient for each token could be binary, based on word count. Run tokenize on sample data and sow the output fordebugging.



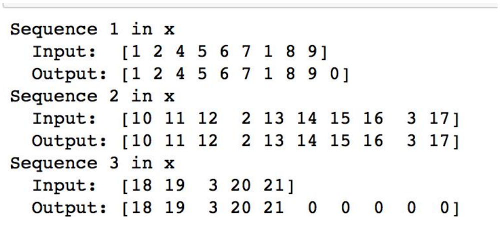
```
{'the': 1, 'quick': 2, 'a': 3, 'brown': 4, 'fox': 5, 'jumps': 6, 'over': 7, 'lazy': 8, 'dog': 9, 'by': 10, 'jove': 11
, 'my': 12, 'study': 13, 'of': 14, 'lexicography': 15, 'won': 16, 'prize': 17, 'this': 18, 'is': 19, 'short': 20, 'se
ntence': 21}

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]
```

Fig. 3. Output of debugging.

3.2 Padding (Implementation)

When batching the sequence of word ids together, each sequence needs to be the same length. Since the sentences are dynamic in length, padding is done at the end of sentences to make them the same length. The function used to perform padding is present in Keras as pad_sequences. It takes three parameters which is x- List of sequences, length-Length to pad to and return padded numpy array of sequences. The output of padding is given below.



```
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]
```

Fig. 4. Output of padding.

3.3 Pre-process Pipeline

The Keras preprocess function is used to create preprocess pipeline. It takes two parameters x- Feature List of sequences, y- Label List of sentences and return a tuple of preprocessed x, preprocessed y, x tokenizer, y tokenizer. The Keras's sparse_categorical_crossentropy function requires labels to be in three dimensions.

4 Models

In this work, we have experimented on various neural network architectures which include simple Recurrent Neural Network, simple RNN with embedding, a Bidirectional RNN and an Encoder- Decoder RNN. We have compared the functioning of these four architectures. The final architecture was built to outperform all the four models discussed above.

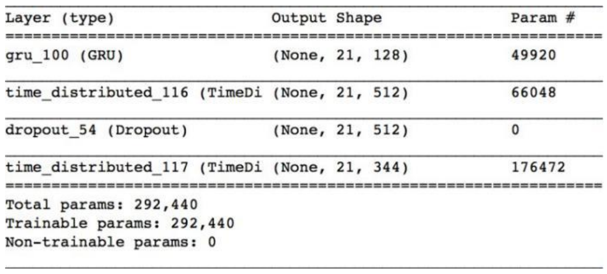
The neural network needs to give the French Translation but it would be translating the input to word ids. The function logits_to_text will bridge the gab between the logits from the neural network to the French translation. It uses tokenizer to turn logits from a neural network to the French translation.

4.1 Simple RNN Implementation

Build and train a basic RNN on x and y. The parameters include a Tuple of input shape, length of output sequence, number of unique English words in the dataset and the number of unique French words in the dataset. It returns a built Keras model.

After this, train the neural network and reduce the batch size to 100 from 1024 and print the prediction. Prior to training the neural network, the layers are built and reshape the input to work with a basic Recurrent neural network.

The information about the layers and parameters are given below in the picture

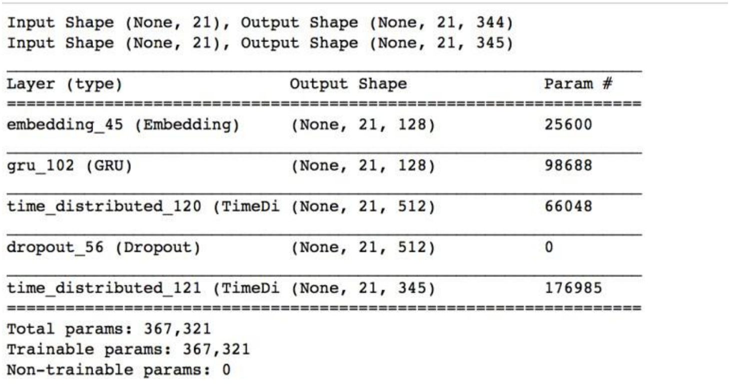


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
gru_100 (GRU)                (None, 21, 128)           49920     
_________________________________________________________________
time_distributed_116 (TimeDi (None, 21, 512)           66048     
_________________________________________________________________
dropout_54 (Dropout)         (None, 21, 512)           0         
_________________________________________________________________
time_distributed_117 (TimeDi (None, 21, 344)           176472    
=================================================================
Total params: 292,440
Trainable params: 292,440
Non-trainable params: 0
_________________________________________________________________
```

Fig. 5. Architecture of a Simple RNN.

4.2 RNN with Embedding

An embedding is a vector representation of the word that is close to similar words in n-dimensional space, where the n represents the size of the embedding vectors. Instead of turning the words into ids, word embeddings are used. The input is reshaped before training the neural network. The passed index length is increased by 1 to avoid index error. After this, the batch size is reduced to 100. The number of epochs taken is 10 and the validation split is 0.2. The prediction is printed.



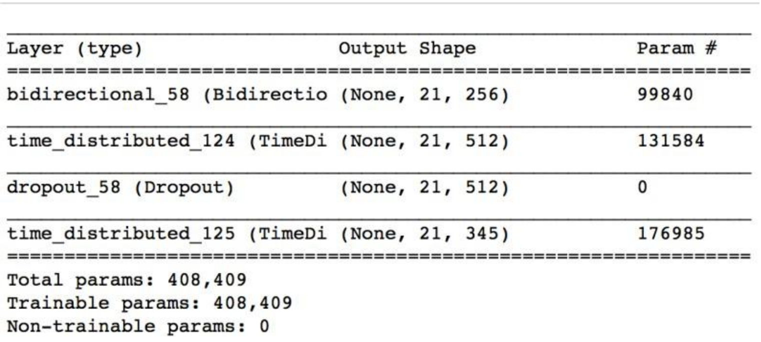
```
Input Shape (None, 21), Output Shape (None, 21, 344)
Input Shape (None, 21), Output Shape (None, 21, 345)

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_45 (Embedding)     (None, 21, 128)           25600
_________________________________________________________________
gru_102 (GRU)                (None, 21, 128)           98688
_________________________________________________________________
time_distributed_120 (TimeDi (None, 21, 512)           66048
_________________________________________________________________
dropout_56 (Dropout)         (None, 21, 512)           0
_________________________________________________________________
time_distributed_121 (TimeDi (None, 21, 345)           176985
=================================================================
Total params: 367,321
Trainable params: 367,321
Non-trainable params: 0
```

Fig. 6. Architecture of RNN with Embedding.

4.3 Bidirectional RNNs

The RNN outputs depend on present inputs and the memory of the previous outputs. But Recurrent Neural Networks don't see the future input. Thus, Bidirectional RNN are used. A Bidirectional RNN is a neural network in which the neurons of a regular RNN are split into two directions- positive time direction and negative time direction. For involving future information, delays need not be added when 2 time directions are incorporated. Implement the Bidirectional RNN by using the tanh activation function.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional_58 (Bidirectio (None, 21, 256)           99840
_________________________________________________________________
time_distributed_124 (TimeDi (None, 21, 512)           131584
_________________________________________________________________
dropout_58 (Dropout)         (None, 21, 512)           0
_________________________________________________________________
time_distributed_125 (TimeDi (None, 21, 345)           176985
=================================================================
Total params: 408,409
Trainable params: 408,409
Non-trainable params: 0
```

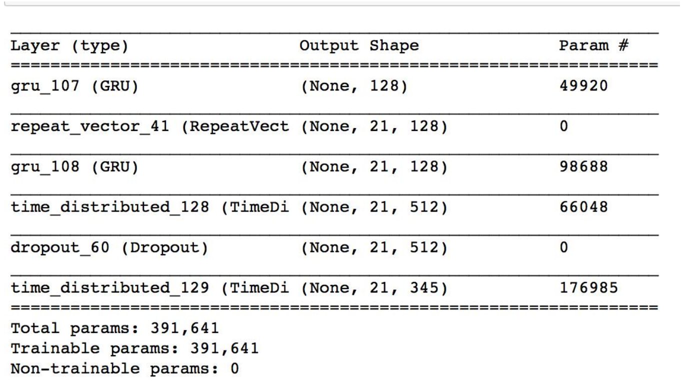
Fig. 7. Architecture of Bidirectional RNN.

4.4 Encoder-Decoder Models

The model consists is made up of an encoder and decoder. The encoder creates a matrix representation of the sentence. The decoder takes this matrix as input and predicts the translation as output. While implementing the encoder reverse the input sequence order for improved accuracy. Design an adapter to fit 2-dimensional output to required 3-dimensional input shape [samples, time steps, features]. The steps involved in encoder model are as follows:

1. Reverse input sequence order for improved accuracy.
2. Adapter to fit 2D output to required 3D input shape [samples, time steps, features]

The decoder is implemented after this. The neural network is trained and prediction is printed.



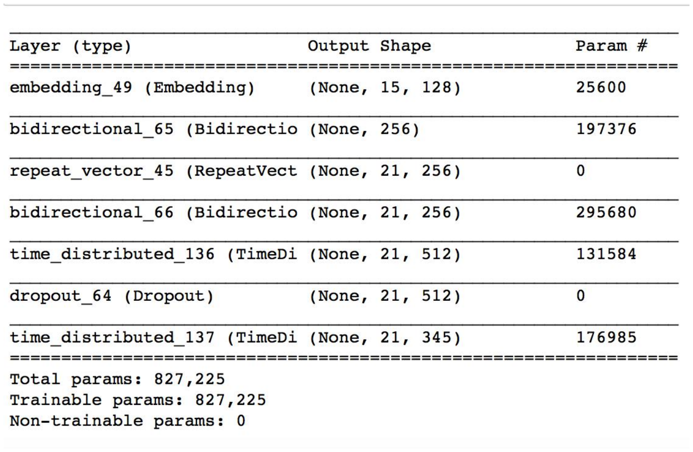
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru_107 (GRU)                (None, 128)               49920
_________________________________________________________________
repeat_vector_41 (RepeatVect (None, 21, 128)           0
_________________________________________________________________
gru_108 (GRU)                (None, 21, 128)           98688
_________________________________________________________________
time_distributed_128 (TimeDi (None, 21, 512)           66048
_________________________________________________________________
dropout_60 (Dropout)         (None, 21, 512)           0
_________________________________________________________________
time_distributed_129 (TimeDi (None, 21, 345)           176985
=================================================================
Total params: 391,641
Trainable params: 391,641
Non-trainable params: 0
```

Fig. 8. Architecture of encoder-decoder model.

4.5 Proposed model

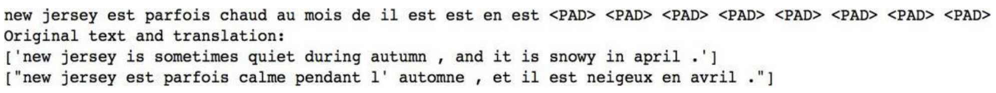
In the proposed model, first embedding is done after which a bidirectional encoder is added. Add an adapter to fit 2D output to required GRU 3D input shape [samples, time steps, features].

Finally implement the decoder. The architecture is given below.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_49 (Embedding)     (None, 15, 128)           25600
_________________________________________________________________
bidirectional_65 (Bidirectio (None, 256)               197376
_________________________________________________________________
repeat_vector_45 (RepeatVect (None, 21, 256)           0
_________________________________________________________________
bidirectional_66 (Bidirectio (None, 21, 256)           295680
_________________________________________________________________
time_distributed_136 (TimeDi (None, 21, 512)           131584
_________________________________________________________________
dropout_64 (Dropout)         (None, 21, 512)           0
_________________________________________________________________
time_distributed_137 (TimeDi (None, 21, 345)           176985
=================================================================
Total params: 827,225
Trainable params: 827,225
Non-trainable params: 0
```

Fig. 9. Architecture of the proposed model.

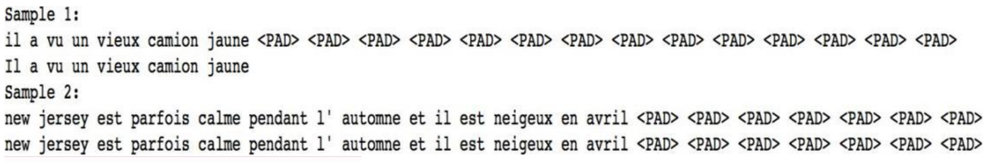


```
new jersey est parfois chaud au mois de il est est en est <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
Original text and translation:
['new jersey is sometimes quiet during autumn , and it is snowy in april .']
["new jersey est parfois calme pendant l' automne , et il est neigeux en avril ."]
```

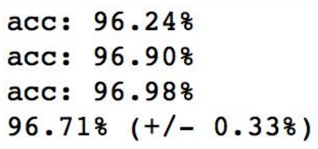
Fig. 10. Original text and translation.

5 Results

The final sentence translations are returned in French with an appreciable accuracy of 96.71% The samples of French translations that were returned is given below;



```
Sample 1:
il a vu un vieux camion jaune <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
Il a vu un vieux camion jaune
Sample 2:
new jersey est parfois calme pendant l' automne et il est neigeux en avril <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
new jersey est parfois calme pendant l' automne et il est neigeux en avril <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
```



```
acc: 96.24%
acc: 96.90%
acc: 96.98%
96.71% (+/- 0.33%)
```

6 Conclusions

Thus, we have built a deep neural network that functions as part of an end-to-end machine translation pipeline to convert English text as input and return the French translation. The proposed network performed far better than the other architectures discussed previously in terms of the validation loss. The accuracy was 96.71%. A random seed was fixed for reproducibility, a 3-fold cross validation test harness was determined and the model was compiled and evaluated.

References

1. Keras documentation
2. SB Sall on “Example based machine translation using natural language processing” 2013
3. Gurleen Kaur sidhu on “Role of Machine Translation and Word sense Disambiguation in Natural Language Processing” 2013
4. MV Reddy on “NLP Challenges for machine translation from english to Indian languages”
5. Kehai Chen, Tiejun Zhao on “A neural Approach to source dependance based Context model for Statistical Machine Translation” 2018