Machine translation using natural language processing

Shiv kumar Mishra , Ishan kumar
School of computer science and engineering
Lovely professional university, India

Abstract. The translation of text or voice by a computer without the assistance of a person is known as machine translation. It is a hotly debated subject in research, and several approaches—including rule-based, statistical, and example-based machine translation—have been developed. Machine translation has advanced thanks to neural networks. The construction of a deep neural network that serves as a component of an end-to-end translation pipeline is covered in this research. Once finished, the pipeline would take in English content and output the French translation. The project is divided into three primary components: pre-processing, model building, and model application to English text.

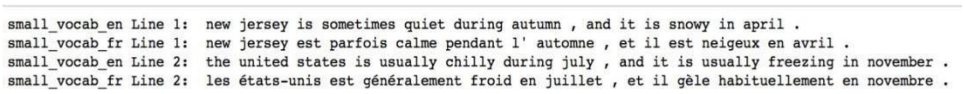
Introduction

Computational linguistics (CL) studies the use of software to translate text or voice from one language to another. One subfield of this research is machine translation (MT). Machine translation only replaces words from one language with words from another, but this does not guarantee accurate translation. Using statistical and neural techniques is a more advanced approach that is also a burgeoning topic to handle the problem of multiple phrase recognition. There is no human interaction in this text translation from one language to another; a computer is handling the conversion. Rule-based, statistical, and neural machine translation systems are the three different categories. One common approach is rule-based, which combines language, grammar, and the support of dictionaries The creation of a complete machine translation pipeline is the main goal of this endeavour. In order to create a more potent system for machine translation from English to French, we have reviewed a number of current designs and have ultimately presented a hybrid approach.

1 Part 1: Dataset

Investigating the Dataset that will be used to train and test the pipeline is the first step in putting any Deep Learning project into action. The datasets most frequently used for machine translation are from WMT, a website devoted to statistical machine translation research. The vocabulary in the dataset is limited by temporal limitations. This enables training to occur in a realistic amount of time.

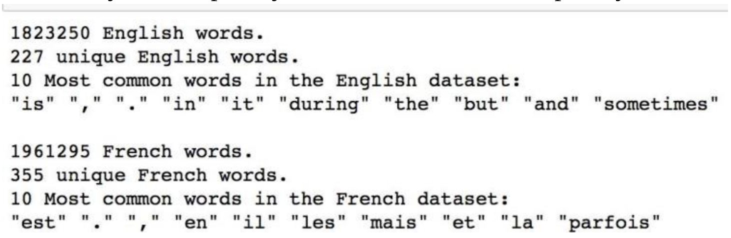
A file path’s contents are loaded. The collection includes translations of English sentences into French. Activate these files to load the data. From each file, print the first two lines.



```
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 the first two lines of each file in English and French.

The punctuation has been separated by spaces in the above graphic. Lowercase has already been converted. The problem's complexity is determined by the vocabulary's complexity. The dataset's complexity is listed 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.

2 Part 2: Pre-process the data

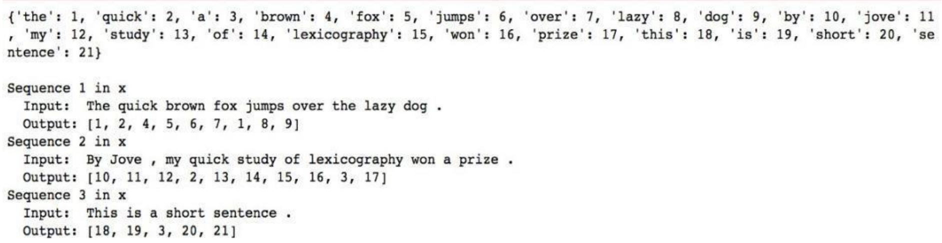
The model does not use the text data as an input. The pre-process procedures of tokenization and padding addition turn the text into collections of numbers.

2.1 Tokenize (Implementation)

In order for the neural network to make predictions about text data, it must understand the input data. ASCII characters are understandable by the network. A series of ASCII character encodings make up text data with the word "dog" in it. Text is transformed into numbers as a result of the network operations conducted, which include multiplication and addition.

The numbers that are given to each character and word are known as word IDs and character IDs, respectively. Word ids were employed in a word level model to produce text predictions for every word. When compared to character level models, this is less complicated.

Tokenizer is a utility class in Keras. This class enables the vectorization of text corpora by converting each text into either an array of integers or a vector with a binary word count as the coefficient for each token. Tokenize some test data, then print the results for debugging.



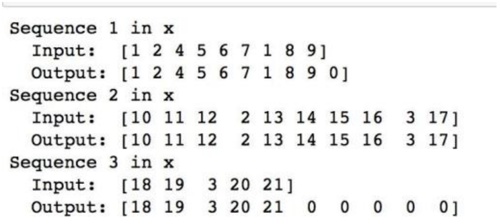
```
{'the': 1, 'quick': 2, 'a': 3, 'brown': 4, 'fox': 5, 'jumps': 6, 'over': 7, 'lazy': 8, 'dog': 9, 'by': 10, 'jove': 11
, 'my': 12, 'study': 13, 'of': 14, 'lexicography': 15, 'won': 16, 'prize': 17, 'this': 18, 'is': 19, 'short': 20, 'se
ntence': 21}

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.

2.2 Padding (Implementation)

Each sequence of word ids must be the same length in order to group them together. The sentences are padded at the end to make them the same length because their length is dynamic. The padding function is available in Keras and is called pad sequences. It requires three inputs: x, a list of sequences, length, the length to pad the array to, and a NumPy array of padded sequences. Below is a list of padding's results.



```
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.

2.3 Pre-process Pipeline

Pre-process pipelines are made using the Keras pre-process function. There are two requirements. A tuple of pre-processed x, pre-processed y, x tokenizer, and y tokenizer is returned when x-Feature List of Sequences and y-Labeler List of Sentences are input. Labels must be in three dimensions in order to use the sparse_categorical_crossentropy function of the Keras.

3 Models

3.1 In this study, we tested a number of neural network topologies, including a bidirectional RNN, an encoder-decoder RNN, a basic RNN with embedding, and a recurrent neural network. These four architectures' operations have been contrasted. The ultimate architecture was created to perform better than the four types mentioned above.

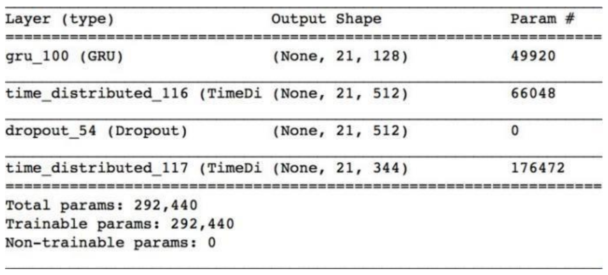
3.2 The input would be converted to word ids by the neural network, but it would also need to provide the French translation. The neural network's logits and the French translation will be connected via the function logits_to_text. Tokenizer is used to convert the neural network's logits into the French translation.

3.3 Simple RNN Implementation

Create and train a simple RNN using x and y. A Tuple of input shape, the length of the output sequence, the number of unique English words and the number of unique French terms in the dataset are among the parameters. A constructed Keras model is returned.

After that, print the forecast and train the neural network with a batch size reduction from 1024 to 100. The layers are constructed, and the input is reshaped to operate with a fundamental recurrent neural network before the neural network is trained.

Below is an image with information about the layers and settings.

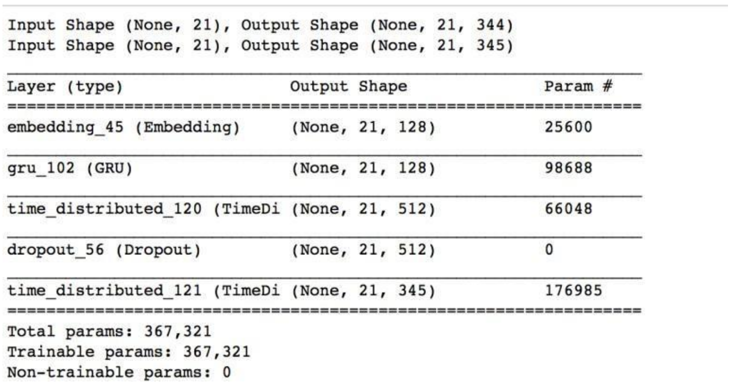


```
Layer (type)                 Output Shape              Param #
=================================================================
gru_100 (GRU)                (None, 21, 128)           49920
_________________________________________________________________
time_distributed_116 (TimeDi (None, 21, 512)           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.

3.4 RNN with Embedding

In n-dimensional space, where n is the size of the embedding vectors, an embedding is a vector representation of the word that is near to other words with a comparable meaning. Word embeddings are employed rather than converting the words into ids. Before the neural network is trained, the input is modified. To prevent index errors, the passed index length is raised by 1. The batch size is then decreased to 100. The validation split is 0.2 and there are 10 epochs used. The Forecast is printed:-

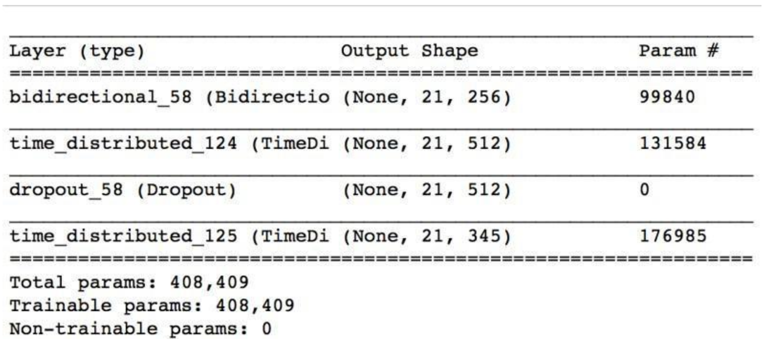


```
Input Shape (None, 21), Output Shape (None, 21, 344)
Input Shape (None, 21), Output Shape (None, 21, 345)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
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.

3.5 Bidirectional RNNs

The RNN's outputs are based on its recollection of past outputs and current inputs. Recurrent neural networks, however, are blind to future input. Bidirectional RNN are utilised as a result. The neurons in a normal RNN are divided into two directions, positive time direction and negative time direction, to create a bidirectional RNN. When two-time directions are included, delays are not necessary to include future information. Utilise the tanh activation function to implement the bidirectional RNN.



```
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
```

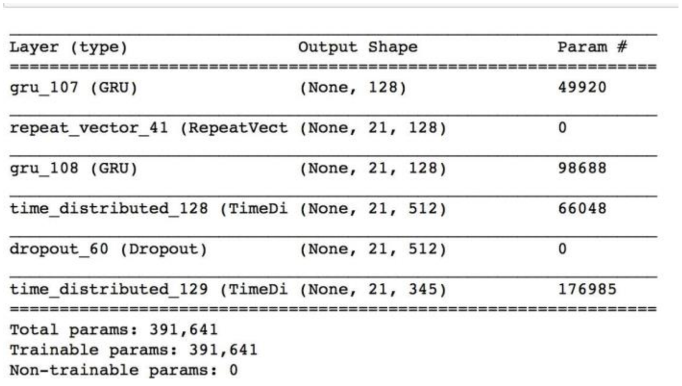
Fig. 7. Architecture of Bidirectional RNN.

3.6 Encoder-Decoder Models

Encoder and decoder components make up the model. A matrix representation of the sentence is produced by the encoder. This matrix serves as the input for the decoder, which outputs a translation prediction. For better accuracy while using the encoder, reverse the order of the input sequence. Create an adaptor that match a desired 3-dimensional input shape (samples, time steps, features) to a 2-dimensional output. The following are the steps in an encoder model:

1. To enhance accuracy, reverse the order of the input sequence.

2. Adapter to match needed 3D input form (samples, time steps, characteristics) to 2D output After that, the decoder is put into use. A forecast is printed after the neural network has been trained.



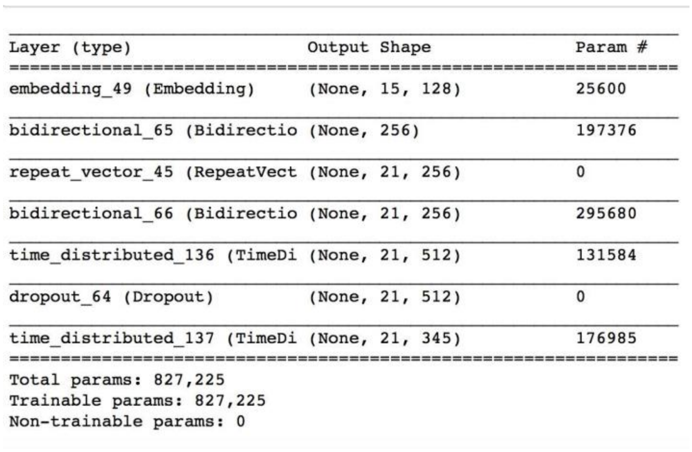
```
Layer (type)                 Output Shape              Param #
=================================================================
gru_107 (GRU)                (None, 128)               49920
_________________________________________________________________
repeat_vector_41 (RepeatVect (None, 21, 128)           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.

3.7 Proposed model

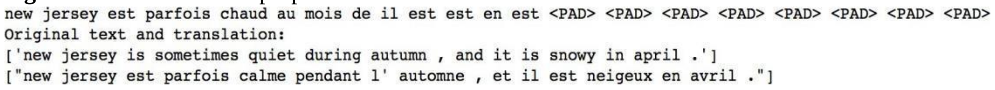
In the suggested paradigm, embedding is completed first, and then a bidirectional encoder is included. To convert 2D output to the appropriate GRU 3D input form (samples, time steps, features), add an adaptor.

Finally, put the decoder into action. Below is a description of the architecture.



```
Layout (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.

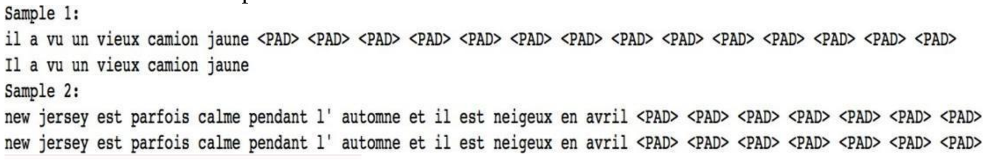


```
new jersey est parfois chaud au mois de il est est en est <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD> <PAD>
Original text and translation:
['new jersey is sometimes quiet during autumn , and it is snowy in april .']
["new jersey est parfois calme pendant l' automne , et il est neigeux en avril ."]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

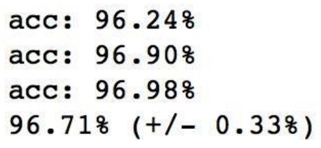
Fig. 10. Original text and translation.

4 Results

The final translations are given back in French with a respectable accuracy of 96.71 percent. Below are some examples of the returned French translations.



```
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%)
```

5 Conclusions

As a result, we developed a deep neural network that works as a component of a machine translation pipeline from beginning to finish, converting English text as input and returning the translation in French. In terms of the validation loss, the suggested network outperformed the other designs previously described by a wide margin. There was 96.71% accuracy. A 3-fold cross validation test harness was established, a random seed was fixed for repeatability, and the model was assembled and assessed.

References

a. documentation for Keras

b. SB Sall's presentation on "Example-based machine translation using natural language processing" 2013

c. The "Role of Machine Translation and Word sense Disambiguation in Natural Language Processing" by Gurleen Kaur Sidhu, published in 2013. MV Reddy discusses the "NLP Challenges for Machine Translation from English to Indian Languages"
d. d. "A neural Approach to Source Dependence Based Context Model for Statistical Machine Translation," Kehai Chen and Tiejun Zhao's 2018 article.