An Exploration of the Deep Learning Translation Model

Likhitha Divyani Eda, Dylan Palmer, Debanjan Mukhopadhyay

Department of Computer Science

University of Massachusetts Lowell

Lowell, MA

LikhithaDivyani_Eda@student.uml.edu, Dylan_Palmer@student.uml.edu, Debanjan_Mukhopadhyay@student.uml.edu

Abstract— Translation has grown to be extremely important in today's globalized world. With the rapid increase in information exchange between people of different countries, there has been a constant demand for making a system that would help people from different communities with different dialects communicate effectively. As a result, many researchers and companies like Google have jumped on to solve the problem by creating and improving the machine translation system. This paper will showcase an effective and ready-to-use deep learning model to translate text from one language to another by using a variety of available data in the form of Bank Statements, transcripts from TED Talks, Movie Subtitles, etc, and available open-sourced computing resources. We used an encoder-decoder implementation based on Bahdanau's attention vectors to build the translation pipeline. It has been found that attention vectors used in our model rapidly improve the performance of machine translation systems by increasing the attention weightage on important parts of the text and hence resulting in an effective translation. Despite facing few challenges like complex dataset and lack of quality resources like a larger GPU, we were able to build a good neural network translation model which can translate from Spanish to English with a fair amount of accuracy.

Keywords— Machine translation system, Deep Learning model, Bahdanau's attention vectors, Recurrent Neural Network, Neural network translation model

Introduction

In the last few years, there has been a revolution in machine translation. New and better translation systems are continually being built using deep learning and replacing older systems designed by linguists using decades of research in statistics. With tools getting easier to use, GPUs getting more powerful and the training data more plentiful than ever, one can now build a language translation system using off-the-shelf hardware and software. Hence, this is an attempt to build a deep learning based translation system that can translate text with a high level of accuracy. The question we are trying to investigate is, can a Deep Learning model give effective results for a translation system compared to the older google translation system without a Deep Learning Model? Our goal is to create our own Neural Network Translation Model with high accuracy.

Our motivation to pursue the project is to improve the efficiency of the translation system by using Deep Learning models which improve the accuracy. In today’s world, anyone with data and computing power can build their own translation system using Deep Learning. The data is available for free which means you don’t have to pay Google fees to use the API.

A translation system helps people from different communities with different dialects communicate effectively. This leads to more ideas exchanged in meeting and fewer problems from miscommunication. It is also very helpful when you visit a country having a different official language than yours. Without effective communication with the local people you wouldn’t be able to navigate through a foreign country.

Related Work

Empirical Methods for Language Processing is one of the biggest and longest running conferences. It has been going on since 2006 and consistently produces research papers about machine translation. Another example is Google Translate which supports over 100 languages and continually upgrades their capabilities as they refine their data mining to better receive the data they need for more refinement.

Different Approaches for the Model

There are several open-source models that already exist working on this problem. We found that there are many ways to implement the translation model. It all comes down to what kind of data we use, the types of neural networks we want to implement, and how we decide the input can be translated so that a neural network can understand.

All the previous research shows that an attention mechanism is key for the neural translation model to make accurate results. They also make use of hidden vectors to create memory vectors.

Proposed Approach

The first step in our project is to obtain data which is in parallel corpora. This means that there are two files in different languages that have pairs of sentences such that each sentence has a translation in the other language. These sentences are mapped to the same location in the other file. The OPUS website has a wide collection of data in many different languages from around the world and they have made files in parallel corpora which are available for free. We gathered data from the OPUS [1] website for English to Spanish language pairs.

In order to have a variety in our text for a better training model, we gathered a variety of forms of language from different sources. These sources include international banks, books, movie subtitles, and TED talks. Using this data we would feed it through an existing neural network [3] that we made slight adaptations to. As with all machine translation models, it first preprocesses the data, then tokenizes it so it can be fed into the encoder. The encoder and the subsequent decoder are both Recurrent Neural Networks (RNNs).

Data Preprocessing

To preprocess the data, we first downloaded all of the files from the OPUS website and unzipped them. Next, we concatenated all english files, and all spanish files into two large files. These files were then concatenated together to create a single file as the RNN could only take single file inputs for its preprocessing. Afterwards, we shuffled the corpus so the model learns the data in random order, promoting better results. Further we created a space between a word and punctuation following it and then replaced any characters that are not within the ascii table with a space. Finally we added a <start> and <end> tokens to each sentence to identify the start and end of it.

Translation Pipeline

The first thing to do in our translation pipeline is to tokenize and pad the data set. Padding is needed because sometimes the input size does not match the output size.

The encoder-decoder implementation is based on Bahdanau's attention which uses attention vectors. The encoder starts with the attention vector. It then creates a context vector. Then the attention weights are given to the context vector to predict the output.

In the training stage, the input is passed into the encoder. The encoder outputs a token which is also the decoder input. The decoder outputs the predictions and the hidden state. Then the output of the decoder is given back to the model to calculate the loss. Finally we calculate the gradients and apply them to the optimizer to go ahead with backpropagation. This is what happens in a single epoch of the translation model. In each epoch we use smaller batch segments for faster and less complex calculations.

Discussion of Results

Fig. 1 below shows a snapshot of the output from one of the runs taken using the tensorflow model [3]. The model takes small batches of the data and goes

through multiple runs until the entire section of input data is fed through it.

```
Epoch 1 Batch 0 Loss 4.7352
Epoch 1 Batch 100 Loss 2.1861
Epoch 1 Batch 200 Loss 1.8133
Epoch 1 Batch 300 Loss 1.7599
Epoch 1 Loss 2.0252
Time taken for 1 epoch 2230.62 sec

Epoch 2 Batch 0 Loss 1.5629
Epoch 2 Batch 100 Loss 1.3789
Epoch 2 Batch 200 Loss 1.3724
Epoch 2 Batch 300 Loss 1.2798
Epoch 2 Loss 1.3686
Time taken for 1 epoch 586.11 sec

Epoch 3 Batch 0 Loss 1.1666
Epoch 3 Batch 100 Loss 0.9290
Epoch 3 Batch 200 Loss 0.8700
Epoch 3 Batch 300 Loss 0.8554
Epoch 3 Loss 0.9387
Time taken for 1 epoch 604.30 sec

Epoch 4 Batch 0 Loss 0.5573
Epoch 4 Batch 100 Loss 0.5633
Epoch 4 Batch 200 Loss 0.5774
Epoch 4 Batch 300 Loss 0.5742
Epoch 4 Loss 0.6204
Time taken for 1 epoch 606.42 sec

Epoch 5 Batch 0 Loss 0.3759
Epoch 5 Batch 100 Loss 0.4407
Epoch 5 Batch 200 Loss 0.3327
Epoch 5 Batch 300 Loss 0.4319
```

Figure 1: The results of the Training Model

Equations

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$

Figure 2: Neural Network Equations

This is the equation of forward pass of a recurrent neural network that maps an input sequence to an output sequence of the same length. The total loss for a given sequence of **x** values paired with a sequence of **y** values would then be just the sum of the losses over all the time steps.

$$W \leftarrow W - \alpha \frac{\partial L}{\partial W}$$

Figure 3: Weights

Loss, **L** is the negative log-likelihood of the true target ***y(t)*** given the input so far**.** Similar to normal back-propagation, the gradient gives us a sense of how the loss is changing with respect to each weight parameter. We update the weights W to minimize loss with the following equation. The process is called backpropagation with time.

Forward pass and backpropagation are the backbones of our neural translation system as it facilitates the training process of the model.

$$
\begin{aligned}
\alpha_{ts} &= \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} && \text{[Attention weights]} && (1) \\
\boldsymbol{c}_t &= \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s && \text{[Context vector]} && (2) \\
\boldsymbol{a}_t &= f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_c[\boldsymbol{c}_t; \boldsymbol{h}_t]) && \text{[Attention vector]} && (3) \\
\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) &= \begin{cases} \boldsymbol{h}_t^\top \boldsymbol{W} \bar{\boldsymbol{h}}_s & \text{[Luong's multiplicative style]} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_1 \boldsymbol{h}_t + \boldsymbol{W}_2 \bar{\boldsymbol{h}}_s\right) & \text{[Bahdanau's additive style]} \end{cases} && && (4)
\end{aligned}
$$

Figure 4: Encoder - Decoder functions

The equations in Fig. 4 describe the functionality of the Encoder and Decoder. The variable **h** seen in the figure represents the encoder output. The variable **W** represents the fully connected layer. The **h bar** represents the hidden state. The variable $\mathbf{a_{ts}}$ represents the attention weights. In the coding aspect, the *attention weights(*$\mathbf{a_{ts}}$*)* are the softmax(score, axis =1). The *context vector(*$\mathbf{c_t}$ *)* is equal to sum(attention weights * EO, axis = 1). The *attention vector(*$\mathbf{a_t}$**)** consists of results of the activation function applied on the **spanish input**. The score is calculated using the Bahdanau's additive style as described in the Proposed Approach section.

Challenges

Despite the work put into setting up the OPUS dataset, it was too complex to be used with the hardware available to us. Even with Google Colab (even pro), The virtual machine ran out of resources and crashed during the execution. This happened no matter the sample size or the batch size of the model. As the main file ended up being 10 GB, we even tried random sampling the file with some head and tail commands combined with some shuffling, to get a smaller file to run through, then removing duplicates,

but even that crashed both hardware and virtual machine.

The other major problem, and the reason we switched to using Google Colab to begin with is that for some reason we could not get the RNN to run on the graphics card. Despite having tensorflow-gpu installed and Anaconda recognizing the graphics card installed, the RNN would continue to run on CPU.

Data Gathered

While setting up the OPUS dataset, and after finding out that it wouldn't work, we used the dataset supplied by the original article to start generating results.

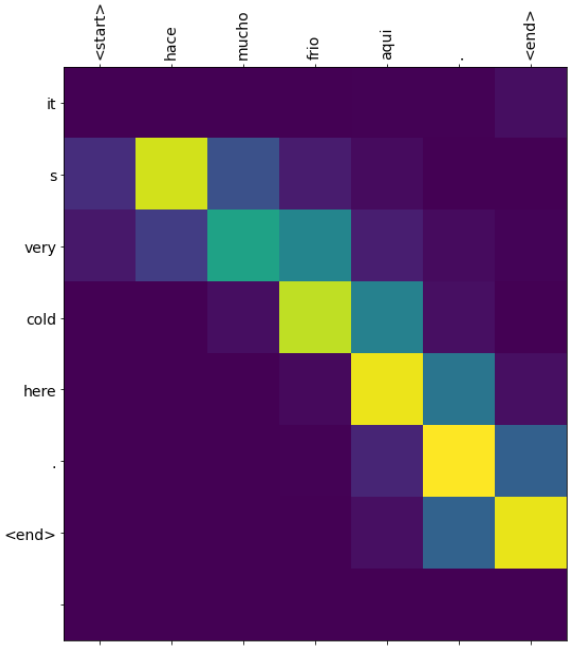

Figure 5: Basic run without changes

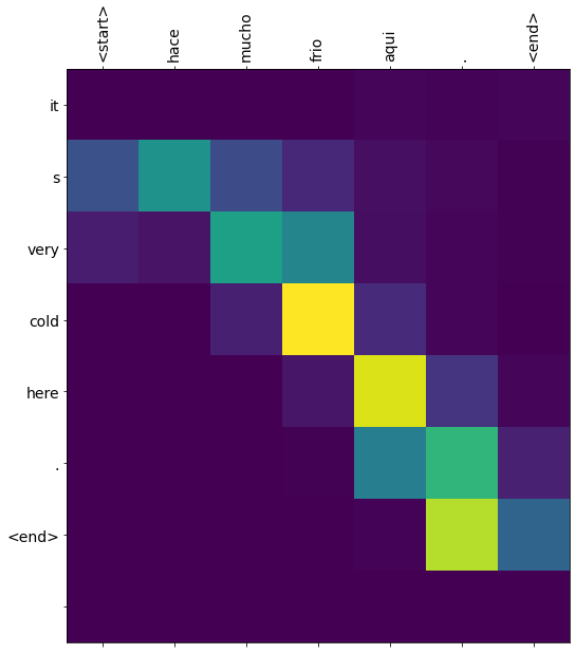

Figure 6: Increase sample size

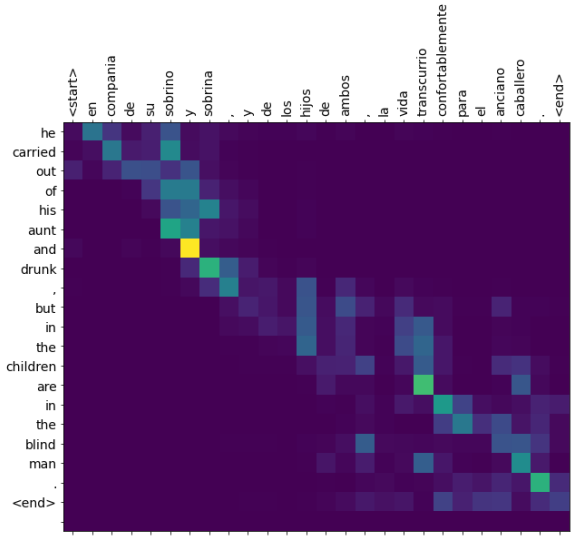

Figure 7: Increased sample size and Epochs

We ran multiple tests that each changed some factors to try and raise the accuracy of the model. The base test ended up with a 9.9% error rate. Fig. 5 shows a run in which doubled the sample size from the base, as well as used an Long Short Term Memory (LSTM) hidden layer. Despite these changes, the error rate only dropped to 8.9%. After getting access to colab pro, we ran a few more tests. Fig. 6 shows the results from that. With five times the sample data, 20 epochs, larger batch rate, and an optimized LSTM,

the error rate got down to 1.6%. This rate is low enough that we were worried about overfitting though, so we did more tests of the translation and found the model was overfit. We ran multiple trials but in the end we couldn't find the correct balance between over and under fitting.

Conclusions and Future Work

In conclusion, we successfully built a neural network translation model which can be trained to translate from spanish to english. We found that an increased number of epochs led to a lower error rate, we never found a good balance between underfitting and overfitting. The algorithms worked better for the datasets which are less complex (word to word vs sentence to sentence translation). Therefore, the model was not at a point to function as well as a google translation algorithm which can take paragraphs and can translate them. Ofcourse, the google algorithm has been built using an abundance of data gathered over the years. The lack of better resources limited our ability to run a better dataset. Despite reaching a low error rate, the translation was pretty spotty (when outside simple sentences). It was hard to tell if that was from overfitting or from lack of dictionary though.

In our future work, we can try to train different data sets for other languages. For example, french to spanish. Given more time, we could also focus on modifying the code so that it can make translations for more complex data. We can also experiment on different neural networks to see which one has a faster functionality and efficiency.

Acknowledgment

We acknowledge those who have done this research previously and gave us the knowledge to implement a neural translation model.

References

[1] "... the open parallel corpus," OPUS. [Online]. Available: https://opus.nlpl.eu/. [Accessed: 12-Apr-2021].

[2] A. Geitgey, "Build Your Own 'Google Translate'-Quality Machine Translation System," Medium, 24-Sep-2020. [Online]. Available: https://medium.com/@ageitgey/build-your-own-google-translate-quality-machine-translation-system-d7dc274bd476. [Accessed: 10-Apr-2021].]

[3] Google, "Neural machine translation with attention : TensorFlow Core," TensorFlow. [Online]. Available: https://www.tensorflow.org/tutorials/text/nmt_with_attention. [Accessed: 12-Apr-2021].

[4] Q. Lanners, "Neural Machine Translation," Medium, 07-Jun-2019. [Online]. Available: https://towardsdatascience.com/neural-machine-translation-15ecf6b0b. [Accessed: 10-Apr-2021].