# Recurrent Neural Networks for Machine Translation

Johnathan Kao - jmk077000
Kendal Wiggins - kdw180003
Matthew Standeven - mts180001
Ryan Bell - rtb210000

The University of Texas at Dallas

**Abstract - Recurrent neural networks (RNNs) are an advanced type of neural network that are characterized by their ability to maintain internal state or "memory" from previous inputs that makes them strong at handling sequential forms of data. This unique property makes RNNs well suited for many NLP tasks which require the context of a word in a sentence in order to accurately predict or generate subsequent words. In order to achieve this form of memory, RNNs essentially stack simpler neural network architectures in sequence, passing parameters from one neural network to the next in a process called backpropagation through time (BPTT). This form of training is a very time consuming endeavor, especially when applied to tasks that involve a large corpus of text and is best handled with parallel processing approaches such as those enabled by the Spark, MapReduce, and Hadoop frameworks. In the following paper, we explore training a RNN with a vanilla architecture and a single hidden layer for the task of machine translation using a parallelized and distributed approach by implementing our RNN and processing within the Spark and MapReduce frameworks.**

## I. INTRODUCTION

Machine learning is about predicting the future by analyzing past data. It can be used for a variety of tasks, including classifying potential software test cases for pass and fail., recommending videos to watch based on other user's screen time. understanding speech and converting it to text or outperforming players in chess. Machine learning algorithms work by detecting patterns in data. These patterns help the system make predictions about future data or possible outcomes. For example, a machine learning algorithm could be used to predict which software test cases are likely to fail, based on data from previous test cases [1]. Machine learning is a powerful tool that can be used for a variety of tasks. However, machine learning algorithms are dependent on the training data and its arguments. Without precaution, the algorithm may overfit or underfit its predictions.

Machine translation is a popular topic in machine learning because it has the potential to revolutionize the way we communicate with each other. Machine translation involves translating natural languages by implementing machine learning algorithms. Instead of using a one-to-one word translation, the purpose of creating a machine translation model is to communicate the original response to a particular outcome. The algorithm does translation by analyzing text elements and recognizing how words influence one another. Popular tools such as ChatGPT, have utilized machine learning with AI to create responses from human natural language. Machine translation can also be used to translate documents, websites, and other forms of content, making it a valuable tool for businesses and organizations that operate in multiple countries. However, there are some limitations to machine translation accuracy. Just like with other machine learning algorithms, machine translation models can sometimes produce inaccurate or even nonsensical translations, and they can be difficult to use with complex or technical language. Nevertheless, machine translation is still great for providing text-to-text services for people.

Machine learning algorithms can also involve big data frameworks, systems that handle large and complex data sets using distributed computing and parallel processing. Big data frameworks can help machine learning algorithms to scale up, improve performance, and handle different types of data sources. Some examples of big data frameworks that are used for machine learning are Apache Spark and Hadoop with MapReduce. MapReduce is an implementation for processing and generating big data sets on clusters with map and

reduce functionality. MapReduce can handle very large data sets that may not fit in memory or on a single machine and can exploit the parallelism and scalability of distributed computing, speeding up the computation and reducing the communication overhead. Machine learning is able to utilize big data frameworks to optimize performance for data analysis.

This document covers our machine translation model made from scratch that implements MapReduce. The paper goes in depth about RNNs and methods our team used to perform machine translation with MapReduce. Afterwards, the outcome and results of our model are covered. Finally, the document will end with a conclusion and give our team's opinions on implementing the machine translation model with our dataset.

## II. BACKGROUND AND CONCEPTUAL STUDY

One of the methods to implement machine translation into a program is to set up a recurrent neural network (RNN) algorithm. RNNs are a type of artificial neural network (ANN) that focuses on anticipating the future of an outcome. RNNs are trained with forward propagation, backward propagation, and gradient descent. In order to make predictions, A neural network is trained and updated with forward propagation, backward propagation, and gradient descent. In forward propagation, neural networks make a prediction based on the input dataset. ANNs are composed of perceptrons. A perceptron consists of neurons (nodes) that take a connection of inputs and their weights, compute the weighted sum, and produce an output. Perceptrons are trained using Hebb's rule - the connection weight between two neurons is increased if both neurons have the same output. Machine learning neural networks take a variant of this rule: if a neuron outputs a wrong prediction, the connection weights from the inputs are reinforced. Neurons are stored inside layers to form a neural network. Each layer communicates to each other by taking the input of the previous layer and producing an output for the next layer. The prediction is made once the final layer produces an output. Backward propagation focuses on updating the weights of the neurons. To minimize the differences between predicted and actual output, back propagation calculates a gradient from the weights in the network. Finally, in order for the model to have the best prediction of the dataset, gradient descent is used to optimize the weights. Gradient descent is an iterative algorithm that adjusts the weights of the neural network based on the gradients from back propagation. The process of training a neural network can be computationally expensive, but it can be very effective for machine translation.

RNNs are used in time-series predictions and machine translation. RNNs contain recurrent neurons. Unlike other neural networks which have a unidirectional process from input to output, recurrent neurons send the output back to itself. This allows RNNs to remember information from previous inputs, which is important for tasks such as machine translation. Each recurrent neuron has weights for the inputs and outputs from previous time steps. An RNN model iterates through time steps - at each time step, the recurrent neuron receives the input and its own output from the previous timestep. Compared to other types of ANNs, RNN can work with a sequence of arbitrary lengths rather than a fixed set of inputs.

Recurrent neural networks (RNNs) are superior to other neural network algorithms in a few ways. First, they can process variable-length inputs, making them more flexible and adaptable to different data formats and domains. Second, they are able to identify complex and temporal patterns in sequential data, making them ideal for tasks such as machine translation and speech recognition. However, like many neural network algorithms, RNN could suffer from the exploding and vanishing gradient. In backpropagation, gradients are calculated by propagating the error from predicted and actual outputs. However, the produced gradients can suffer from indistinct or dramatic change. The vanishing gradient problem happens when the weights don't change due to gradients reducing in size. In contrast, exploding gradients happen when the weights change drastically - the weights become so large that the algorithm diverges. To counter the vanishing and exploding gradients problem, RNNs make use of long short-term memory to control the flow of information and gradients through time.

## III. METHOD

### *A. Data*

We are using the alignment-en-fr.txt dataset in the NLTK comtrans module for our project. This dataset contains 33,334 English and French sentence pairs that represent English to French translations completed by a domain expert and is suitable for machine translation training for a ML model.

### *B. Preprocessing*

Preprocessing the training dataset properly is crucial to strong performance for any machine learning model and this is especially the case in NLP tasks such as machine translation which involve additional pre-processing steps for handling the corpus. In training our model we took three steps to prepare the text data [4]: standardization, tokenization, and vector encoding. First, in the

standardization phase we transformed our data into a format that is more understandable to a machine by making every word lowercase but we did not perform stemming and stop-word removal since these would clearly result in worse machine translations. In many NLP tasks punctuation isn't relevant so it's removed during pre-processing, but this is not the case with machine translation. Punctuation is particularly important for giving some words meaning in French so we need to keep that punctuation in our model. Furthermore our team first decided to shorten the sentence length to 10 words maximum but due to performance issues, it was further decreased to 6 words. Increasing the sentence length further results in additional time-steps during BPTT which increases the probability of encountering gradient instability such as vanishing gradients.

Next, tokenization is where text is broken down into smaller units or numerical values called tokens where splits can consist of characters, words, or a group of words. After our team tokenized the dataset we assigned each unique word a unique integer which is referred to as 'vocabulary indexing'. The purpose of this step is to create a dictionary from our corpus that can later be used for both one-hot encoding of words and for converting the RNN model's output into word translations. Finally in the last pre-processing step, we convert each word token into a one-hot encoding, meaning a vector with an index for every word in our corpus, using the indices generated during vocabulary indexing. Each input sentence that our RNN model is then trained on is a sequence or list of these one-hot encodings representing each French or English sentence.

### C. *How we've implemented our RNN*

Our program is written into an online Google Colab python file (.ipynb) since Google Colab allows our team to concurrently work together with its concurrent read/write functionality. Additionally, Google Colab allots its users computational resources which our team can adjust according to the needs of our project. The NLTK library and default Python libraries were used for dataset preprocessing. The numpy library was used for implementing the backpropagation through time algorithm and generally for handling the mathematical operations and weight updates of gradient descent.

### D. *How we've implemented MapReduce*

To demonstrate machine learning with big data framework implementation, our team decided to use the Apache Spark library for Python. Spark uses the Hadoop library for MapReduce functionality. MapReduce will be used to perform forward and backward propagation onto the dataset and train the RNN model. Although the mapping and reduce functionality are run in different processes, each functionality has its own concurrent processing power by being able to split data into different chunks and process them in parallel. This allows for faster processing times and more efficient use of resources. By utilizing the Apache Spark library, machine learning can be implemented with a big data framework. In our implementation we run the forward and backward propagation steps on a configurable percentage of the dataset each epoch. For example, if we configure the percentage to be 95% of the dataset then we get 95% of the sentences and run the forward and backward propagation on each sentence individually. The trick to this approach is combining the results of the backward propagation for each sentence so we update our weight matrices with each gradient produced correctly. We do this by computing the gradient for each sentence individually and then adding all the gradients together. Then that total change is added to the weight matrix. In effect, this means that every epoch, or iteration, the weight matrices are updated using the whole dataset at one time. Using this approach we can vastly reduce the amount of time it takes a single epoch to run. However, using this approach we swap micro changes for very large changes which could make it more difficult for the algorithm to converge. This can be mitigated by adjusting the learning rate to compensate.

## IV. RESULTS

Initially, instead of representing the French and English words as one-hot encodings from the vocabulary index we attempted to encode each input as a word2vec word embedding and then attempted to use the word embedding to predict a corresponding word embedding using the RNN model. Word embeddings have superseded the one-hot encoding word representations in popularity over the last decade since word embeddings capture semantic relationships between words and most importantly have drastically reduced dimensionality compared to one-hot encodings. However, the model's representational ability using only a single hidden layer was far too weak to represent such a complex vectorization relationship using only a linear activation function. Additionally our team discovered that the model was generating outcomes randomly since we weren't able to find an effective activation function for handling this type of word2Vec vector error comparison with our architecture. Without having a coherent loss function the weights in the model shifted erratically. The encoding of the vectors

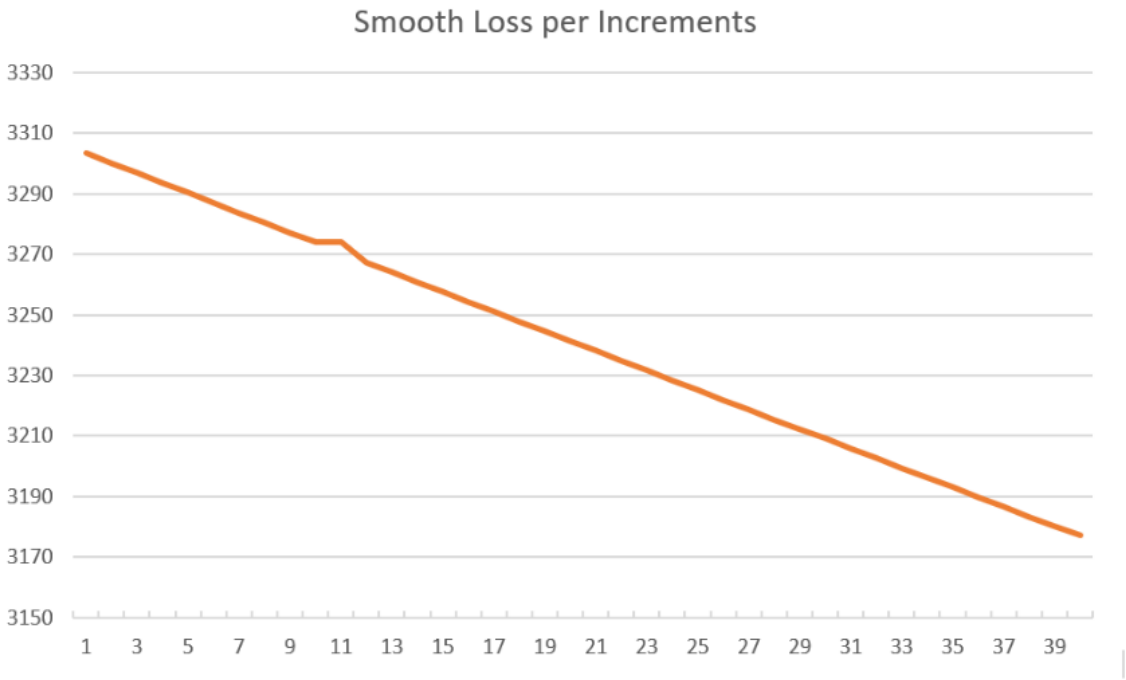


Figure 1: Smooth Loss Value after each iteration from training the RNN model

needed to be something the model could replicate. Our team eventually concluded that we need to change to preprocessing the text with one-hot encoding representations instead.

After switching to the one-hot encoding representation we encountered another problem. Running the model using MapReduce means that every iteration we run every sentence through the model at one time. Using one-hot encoding produces input vectors that are each the size of the entire vocabulary for both the input and the output which resulted in vectors that were very large. Using Google Colab puts major constraints on the amount of resources we have available so we found that we would initially run out of memory using this approach. NLP tasks such as machine translation are especially prone to this problem and eventually we needed to reduce the size of the sentences significantly in order to allow our program to run in-memory on Google Colab. This was the primary deciding factor in making the maximum sentence size allowed to be 6.

In evaluating our RNN model, we chose to evaluate the model's predictions ourselves by manually comparing the English translations with the labels in the dataset and using our natural understanding of the English language to determine how good the translation was. This may seem to be an incorrect approach as the evaluation metric is not quantitative but there is currently no better metric than using a domain expert (a human that knows one or both languages) to assess the model's translations in machine translation although alternative systems such as the BLEU score do exist. While we did not have sufficient time to extensively test the model, it is clear that the RNN model is working correctly in the sense that the model improves in its ability to perform machine translation over time. For example, in one case when translating 'the debate is closed' the model's initial prediction was 'closed debate debate debate' but after 50 iterations of training it predicted 'the then the closed the' achieving an accuracy of 40% in terms of word for word translation compared to an initial accuracy of 20%. In another case, after only 10 iterations the model was able to fully predict all 5 words in 'the debate is closed.' after initially not being able to predict any of the correct words. In other cases the model was not able to predict any words correctly even after 100 iterations of training. The significant differences in translation performance depended primarily on the choice of the hyperparameter settings for the learning rate and the number of neurons in the hidden layer, demonstrating the importance of hyperparameter tuning in the performance of machine learning models. In the future, we would like to further experiment with the hyperparameters and do more extensive testing to determine the best set of hyperparameters. However, since our study is more of a proof-of-concept we did not put emphasis on this at this time. Considering the simplicity of our RNN architecture and the complexity of the best machine translation models in

use today we consider this a good overall performance by the RNN model. Finally, in Figure 1 we can see that the loss is steadily decreasing, indicating that with each iteration of gradient descent the model's machine translation performance is slowly improving.

## V. CONCLUSION

In conclusion, our team created a basic RNN model with a vanilla architecture, a single hidden layer and a one-hot encoding word representation using a MapReduce architecture and were able to achieve basic French-to-English translation capabilities on a machine translation task. Overall, we believe the model performed reasonably considering machine translation is known to be incredibly challenging with strong results only being achieved in the last decade with the development of advanced approaches such as the Seq2Seq and encoder-decoder approaches for language translation [5]. Additionally, a simple RNN with a single hidden layer has minimal representational power compared to deeper neural networks and the word-for-word translation approach is generally considered to be relatively weak. In future work, we plan to improve our model by adding more hidden layers and trying advanced RNN architectures such as long short-term memory (LSTM) and gated recurrent units (GRUs) to see if these are able to perform machine translation better than vanilla RNN architectures. Alternatively, trying to use two neural networks together as in the encoder-decoder approach to machine translation would also be interesting to explore. Finally, our team would be interested in using more computational resources either through the cloud on Databricks or locally using GPUs so that the model could be trained using larger sentences and with fewer performance limitations.

## REFERENCES


[1] C. Wei, Y. Sun, Y. Sheng and S. Jiang, "Machine Learning based Combinatorial Test Cases Ordering Approach," 2021 IEEE International Conference on Software Engineering and Artificial Intelligence (SEAI), Xiamen, China, 2021, pp. 37-42, doi: 10.1109/SEAI52285.2021.9477533.

[2] F. Ois Chollet, *Deep Learning with Python, Second Edition*. Manning Publications, 2022.

[3] A. Géron, *Hands-on Machine Learning with Scikit-Learn, Keras, and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems*. Sebastopol, CA: O'Reilly Media, Inc., 2019.

[4] J. Grus, *Data Science from Scratch: First Principles with Python*. Beijing: O'Reilly Media, 2019.

[5] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in Advances in Neural Information Processing Systems 27 (NIPS 2014), 2014, pp. 3104-3112.