Comparison of Word2vec with Hash2vec for Machine Translation

A project

Presented to

The faculty of Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements of the Class

CS 298

By

Neha Gaikwad

May 2020

The Designated Project Committee Approves the Master's Project Titled

Comparison of word2vec with hash2vec for Machine Translation

By

Neha Gaikwad

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2020

Dr. Christopher Pollett	Department of Computer Science
Dr. Robert Chun	Department of Computer Science
Dr. Thomas Austin	Department of Computer Science

ACKNOWLEDGMENT

I would like to express my sincerest gratitude to Dr. Christopher Pollett for his pertinent guidance, advice, and collaboration throughout the duration of my project.

I consider myself to be extremely fortunate to have had an opportunity to work with someone as brilliant as him.

I am also grateful to my committee members Dr. Robert Chun and Dr. Thomas Austin for providing their valuable feedback and guidance.

Finally, I thank my wonderful family and friends for their countless hours of support and encouragement along the way.

ABSTRACT

Machine Translation is the study of computer translation of a text written in one human language into text in a different language. Within this field, a word embedding is a mapping from terms in a language into small dimensional vectors which can be processed using mathematical operations. Two traditional word embedding approaches are word2vec, which uses a Neural Network, and hash2vec, which is based on a simpler hashing algorithm. In this project, we have explored the relative suitability of each approach to sequence to sequence text translation using a Recurrent Neural Network (RNN). We also carried out experiments to test if we can directly compute a mapping between word embeddings in one language to word embeddings in another language using Linear Regression followed by Principal Component Analysis (PCA).

We trained the word2vec model for 24 hours using google collab default settings. This word2vec model when applied to sentence translation produced results with 85% accuracy. Surprisingly, the hash2vec model performed relatively well with 60% accuracy. The hash2vec model required only 6 hours of processing time which saved a lot of time spent in training the word2vec model. Further research can be carried out using the hash2vec technique on larger datasets and applying it to different machine learning models.

Keywords **- Deep Learning, Recurrent Neural Network (RNN), Principal Component Analysis (PCA).**

Table of Contents

1. INTRODUCTION

One of the important aspects of Machine Translation is to represent words in a suitable form which facilitates applying mathematical models and perform key operations. Vectors and matrices are few such forms where Linear Algebra techniques can be applied to perform operations like shaping the data, finding similarity. This makes these forms suitable to be used in Machine Learning models. A word embedding is a method of transforming words into meaningful low dimensional vectors and is often coupled with the Machine Learning or deep learning models. These models are efficient while working with the data in the form of numbers. Word embeddings are widely used in real-time applications such as Spam Filter [2], Google Translate [7], etc.

Mikolov et al. [3] introduced the Skip-gram model which focused on generating low dimensional meaningful vectors from words. This model was found to be effective when applied to evaluate syntactic and semantic relationships of words. Skip-gram model preserved the contextual information of the words in the transformed vectors which can be validated by visualizing data in the predefined vector space. Mikolov et al. [2] extended their research and successfully showed that employing subsampling techniques in the Skip-gram model results in performance enhancement. They named vectors generated using these models as word2vec which is very popular in the field of deep learning. Google's word2vec library gensim[4] utilizes the Skip-gram model with Neural networks to generate the vectors. Goldberg and Levy [9]

demonstrated that the negative sampling when applied to word2vec produces similar results as the regular word2vec model with much better time complexity. But the time required to generate word2vec vectors was still a couple of days. Yahoo [5] has developed an application known as Spam Filter by implementing the word2vec using the Neural Networks model.

In contrast, Argerich, Cano, and Zaffaroni [1] demonstrated another technique to generate the word embeddings using feature hashing where they applied a hash function to the words in the corpus and tried to represent words in the form of vectors. Weinberger, et.al [8] discussed "large scale multitask learning can be effectively optimized for time and space complexity by applying feature hashing algorithms". They experimented with multiple hashing algorithms to avoid collision and maintain the uniqueness of the features.

The main purpose of this project is to compare the performance of word2vec with hash2vec when applied to Machine Translation. We explored two approaches for doing the text translation, first was to use Recurrent Neural Network (RNN) and use the vectors from word2vec and hash2vec models in the embedding layer of the respective RNN models. This approach yielded very good results while working on sentence to sentence translation problems.

The dataset for the second approach was an ordered bilingual dictionary with respective word2vec and hash2vec vectors. We applied linear regression to map the vectors (language1) to vectors (language2) and predicted the output using cosine similarity. This approach was effective only for the word to word translation. The vectors acquired from the word2vec and hash2vec

techniques during the project execution were also used for other applications such as finding word similarity, language outlier detection. These follow up applied experiments led to promising results. All the results of the above experiments are covered in detail in the later sections of this report.

Text translation is important because not every time a human translator will be available. Text translators enable people to translate a topic in their native language so that they can understand it better. Also, text translation tools and software are emerging in the market to contribute to the global economy. People with different backgrounds, languages can work together, share ideas with the help of language translation. Also, some ancient scriptures can be translated into other languages to study human behavior, thoughts, and cultures in earlier civilizations. This helps archaeologists to understand few patterns and help in the discoveries. Text translations are also useful in spreading information like TED talks to motivate people all over the world. Having such text translation tools handy would attract tourists to visit and explore different nations and their cultures, studies, etc.

Figure 1 shows the high-level flow of the project which includes both word2vec and hash2vec approaches. The rest of the report is organized into three sections such as Background, Design and Implementation, and Conclusion. The Background section deals with providing an in-depth analysis of techniques, architectures that are used in generating word2ved and hash2vec models. It also discusses some of the deep learning models like RNN, sequence to sequence model, and a

few Machine Learning models like Linear Regression, PCA. The Design and Implementation section outlines a summary of technical specifications of the project such as environment, Machine Learning model parameters, datasets used, data preprocessing strategies, hardware requirements, etc. It also compares experimental success metrics for all the techniques employed in this project. The Conclusion section summarizes conclusions and proposes future enhancements to the techniques implemented in this project.

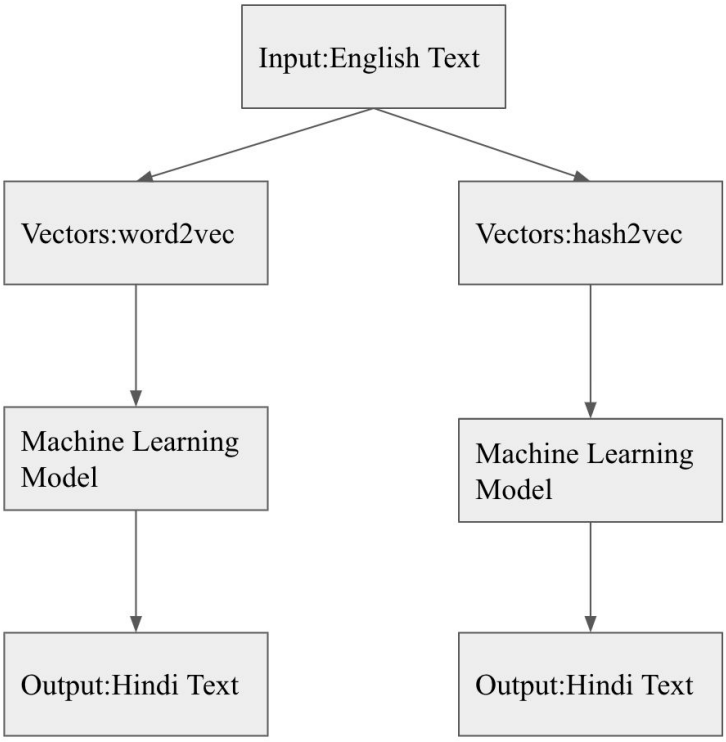


Figure 1. Flowchart of Machine Translation high-level approach

2. BACKGROUND

This section focuses on the technical details of word2vec, hash2vec, Neural Network, deep learning models, and Machine Learning models applied in the project. It is necessary to understand how these models work, to get the complete idea of the project.

Following this, covers the basics of word embeddings and what are word2vec and hash2vec methodologies. Also, how RNN is used with the encoder-decoder technique for Machine Translation.

2.1 NEURAL NETWORK

Neural Network is the type of Artificial Intelligence which works similar to the human brain.

It is widely used in all the fields that involve classification, clustering, predictive analytics, etc.

Neural Networks can be explained with an example of a letter recognition problem. It is a difficult task to recognize the handwriting of different people because different people have different writing styles. Many handwritings with their corresponding representations are fetched into the Neural Networks. These are called training samples. These training samples are then applied to a series of algorithms within the network. The human brain would recognize each character with its unique representation. Similarly, Neural Network learns character recognition from the training samples. Prior research generally confirms that the performance of the Neural Network is directly proportional to the size of training samples. Hence the larger training datasets result in more accurate results from Neural Network.

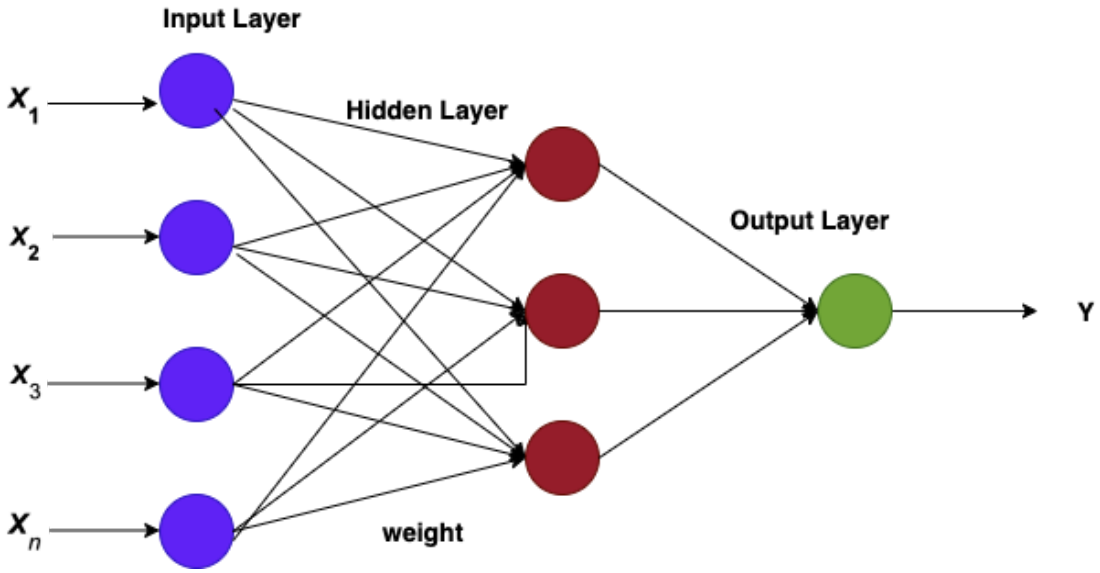


Figure 2. Basic structure of Artificial Neural Network

As shown in Figure 2 Neural Network consists of a web of neurons with multiple layers in the network. Neural Network has multiple inputs with numbers of intermediate hidden layers that are interconnected using neurons. It tries to generate the output that would be as close to the expected output as possible. We will discuss the structure of neurons in detail in the following sub-section. A Neural Network can contain millions of neurons to enhance the accuracy of the network.

Figure 3 depicts the basic structure of a single neuron in the Neural Network. It consists of an input layer, a hidden layer, and an output layer. The data in the neurons are nothing but the

mathematical functions and equations which are the building blocks of the Neural Networks. These functions are used in generating outputs for unseen data.

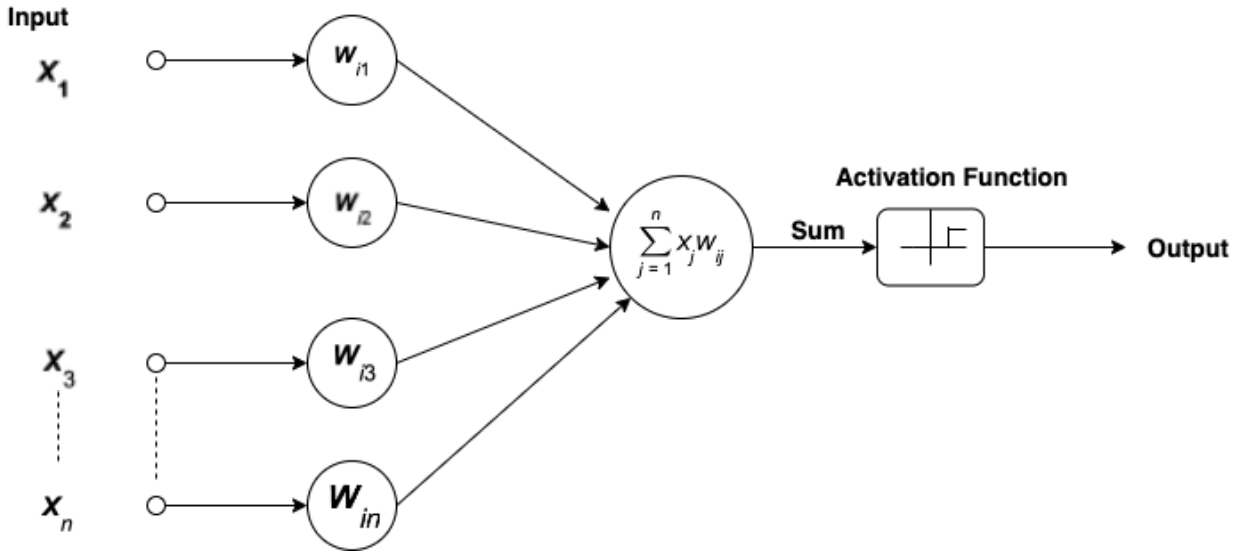


Figure 3. Artificial Neuron structure

Input Layer:

This layer is responsible for the input to the network. This may contain different types of data such as text, image, etc. In our case, it's a text. The above image shows $X_{1,}X_{2,}X_{3,}...X_n$ in the input layer.

Hidden Layer:

This layer consists of multiple sub-layers. As shown in Figure 3, the hidden layer is made up of weight, bias and activation function. The output generation process happens step by step in the hidden layer. In the first step, the input is fed to the neuron which then multiplied by the predefined weights. The second step adds the output from the first step with the bias which is again a predefined component of the network. This sum is then applied to the activation function

which produces the output. This output is then matched with the expected or labeled output. This is a simple training phase of a neuron. There are many parameters and functions to be considered in the hidden layer. Every hidden layer is connected to its previous and next layer, the input has come from the output of the previous layer and output is given as input to the next layer. The previous studies show that the number of hidden layers can affect the accuracy of the Neural Network.

The neuron can be mathematically defined as:

$$Y = X_1 \bullet W_1 + X_2 \bullet W_2 + X_3 \bullet W_3 + \cdot \cdot \cdot \cdot X_n \bullet W_n + bias$$

$X_1, X_2, ...X_n$ are the inputs.

$W_1, W_2, ...W_n$ are the weights.

bias is a constant which is predefined to avoid overfitting.

There are widely known activation functions in Neural Networks which are being used to solve real-time problems such as sigmoid, ELU, RELU, tanh. Sigmoid function gives output in 0 or 1, this is used where the binary decision is needed for whether the given input belongs to the particular domain or not.

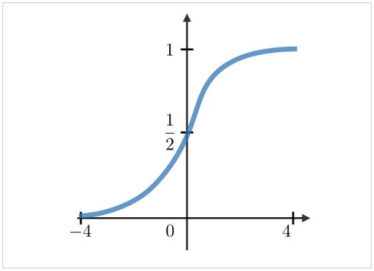
Softmax functions are a type of activation function that is preferred while solving classification problems because they produce output in the form of the probability distribution of the input classes. They are capable of producing results in the binary form.

The equations for sigmoid, tanh and RELU are as below:

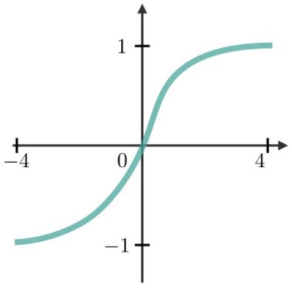
Sigmoid: f(x) = $\frac{1}{1+e^{-x}}$

tanh: $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

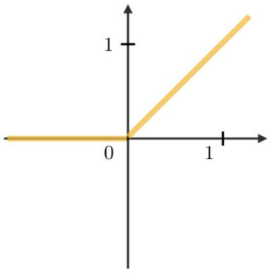
Relu: f(x) = max(0,x)



Sigmoid function



tanh function



Relu function

Figure 4. Activation functions

2.1.1 Gated Recurrent Unit (GRU)

The deep Neural Network consists of many hidden layers which makes the model complicated but it tends to learn complex problems and gives more accurate results. During the training phase, the process of learning from the previous iteration is called backpropagation. In backpropagation, the difference between the output from the previous iteration and the expected output is calculated. This difference is utilized in defining the error gradient. The error gradient is calculated but as there are any number of layers, the error gradient becomes so small for the earlier layers that means the neurons in the previous layers tend to learn slower than the next layer which may degrade the overall performance of the model [21]. This problem is known as the Vanishing Gradient Problem. Relu functions seem to work effectively with rare chances of causing Vanishing Gradient Problems in their network. A Gated Recurrent Unit fixes the problem of the vanishing gradient through the use of two gates: an update gate and a reset gate [12]. These gates or vectors act as gatekeepers that decide about the new information that should be passed to the output of the cell. These gatekeepers use the sigmoid and the tanh function. First, the output of the previous input h_{t-1} and x_t are added together through the use of the following formula:

$$Z_t = \sigma(W(z) \bullet X_t + U(z) \bullet h_{t-1})$$

W, in this case, is the weight of the input, and U is the weight of the previous input. Once added, the output of this sum needs to be between 0 and 1 followed by an application of a sigmoid

function. This gate is the update gate and acts as a way of deciding how much of the past information to use.

The reset gate is used to determine how much of the information the model should forget. This is determined through the use of the following function:

$$r_t = \sigma(W(r) \bullet x_t + U(r) \bullet h_{t-1})$$

There are two outputs O_t and v these would be translated to h. The next step is to determine what to get rid of in the current context. Through the use of the Hadamard product the value output from the reset gate is multiplied to each value in the Weighted h_{t-1} input.

$$h'_t = tanh(W \bullet x_t + r_t \odot U \bullet h_{t-1})$$

The result of this gates is the output ht that through the use of the sum of these previous values shown in the formula here:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t$$

The result of the value h_t is output to the next state.

$$C_t = f_t \bullet C_{t-1} + i_t \bullet C_{t'}$$

2.1.2 Attention

Although models like seq2seq have proven to be reliable models in the world of Neural Networks, there are still issues inherent to the class of natural language processing and neural Machine Translation that need to be addressed. This issue is related to the long sequence translation. Architectures such as seq2seq perform quite well on short sequences, but, through our experiments, the model tended to diverge and produce poor results. A way to fix these issues is demonstrated in [10] using "attention". The concept of attention was introduced by Bahdanau

et al. [11]. This approach introduced another middleman to the architecture that utilized the annotation vector output by the state at each point H ($h_1, h_2, ...h_t$) that is output from each state and computes a context vector that is computed from weights associated with each annotation. The weighted annotation scores are computed from the hidden states of each annotation state. This is parameterized as a feed-forward Neural Network. The expected annotation is the resulting c_i vector over all annotations with probabilities α_i .

2.2 WHAT ARE WORD EMBEDDINGS:

At a high level, it is simply word association with vectors. There are mainly two architectures known as Continuous Bag Of Words (CBOW) and Skip-gram Model which are being widely used to represent words into the vector form [3]. Both architectures require the Neural Network to process the words and convert them into vector form. CBOW and Skip-gram Model are related to the context of the word. CBOW tries to predict the word given the context whereas the Skip-gram model tries to predict the context given the word as input [12]. A One-hot encoding is a technique to convert any text form into a simple stream of numbers. It represents the position of a word as 1 and other words as 0 in the vector. For an instance, consider a sentence "This is the report", this can be represented in the one-hot encoding form as: This = [1,0,0,0] similarly, is = [0,1,0,0], the = [0,0,1,0], report = [0,0,0,1]. To visualize these words into the vector space, we can consider a 4-dimensional space and these words will be presented in the form of unit vectors in this space. The words with similar meanings such as worsen and exacerbate should be placed

near to each other and not at the random position in the space. Hence these numerical representations can be further processed using CBOW or Skip-gram model to consider the context of the word.

2.2.1 CBOW MODEL

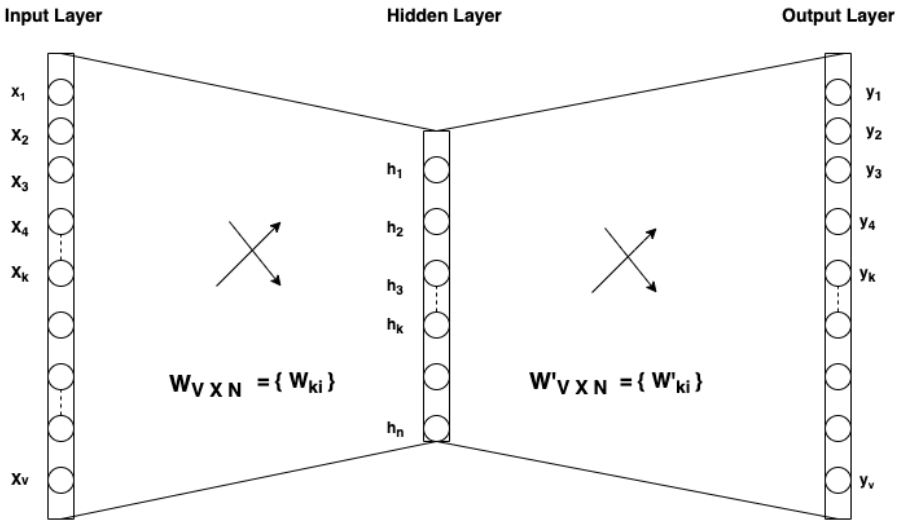


Figure 5. CBOW Model simple architecture

This architecture focuses on the input word and tries to predict the output word according to the context. If we consider a word depression, it could mean a clinical depression or economic depression, how to predict which one is considered. For this purpose, a previous word that might be clinical or economical is considered and then the next word can be predicted.

As shown in Figure 4, if a signal word is considered as an input to the Neural Network, the one-hot encoding of the word is given as input to the Neural Network and its output

representation in the form of one-hot encoding. The error is measured between the output vector and output one-hot encoding vector and accordingly the model learns its relationship with the output vector. The input vector is a vector represented using one-hot encoding of size V. Then the N being hidden layer length and output vector would be of length V with softmax values.

2.2.2 SKIP-GRAM MODEL

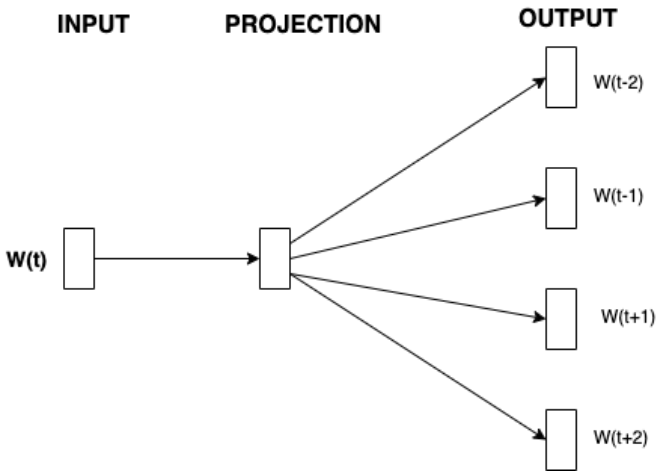


Figure 6. Skip Gram Architecture

This model is the opposite of the CBOW model, where the input layer would contain the target word into a one-hot encoded vector form and the output layer would be having the probability distribution of the context words [3]. As shown in Figure 6, w(t) is the input vector with one hidden layer. The hidden layer will perform the dot product between the weights and the vector and then pass it on to the output layer [3]. Post output layer the softmax activation function is

applied to determine the probabilities of the words occurring in context words at that particular position.

The main difference between the CBOW and Skip-gram model is that the CBOW considers the context and tries to predict the output word, on the other hand, the Skip-gram model takes context word as input and tries to predict the probability distribution of words concerning the context word at that particular position as an output. The architecture preference depends upon the size of the dataset. For the smaller datasets, the Skip-gram model is preferable but for bigger datasets, the CBOW model is preferred [3]. As per [5], the Skip-gram model considers all words in the same context and then tries to find the output in the form of probability distributions. When it comes to rare words prediction, the Skip-gram model seems to work better than the CBOW model. Since the CBOW model tries to output a single word, it requires more training as compared to the Skip-gram model.

2.3 HASH2VEC

Argerich et. al [1] proposed a new technique to create word embeddings using a different approach that did not involve Neural Network, they administered the hashing function to create the vectors representing word embeddings. The CBOW model makes the co-occurrence matrix of size million by millions for a million words. Argerich and colleagues mention that “It becomes impossible to process such a huge matrix and hence in the case of Glove, word2vec the vectors with predefined length were generated to keep this matrix optimized” [1]. Models

involving Neural Networks require stressful and time-consuming training to process millions of words in their respective vectorized structures. However, the Hashing technique does not require any training, it would require some processing time which is negligible as compared to the word2vec training time.

2.3.1 WHAT IS HASHING

Hashing is a technique of taking input of variable length and applying some mathematical function over it and converting it into fixed-length output [17].

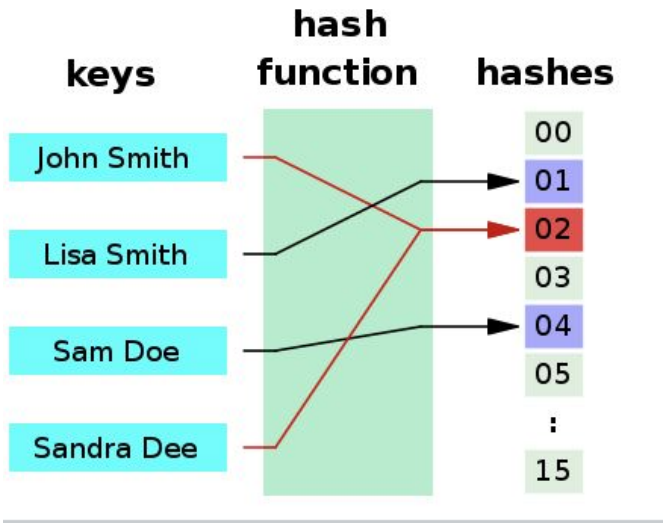


Figure 7. Hashing Technique

Figure 7 shows a simple hashing function applied on keys to map them to the integer values. The hash function is nothing but a mathematical function that can process the input and transform it

into some form used as a value. Hashing has helped in many areas such as information security, code optimization, word processing, etc. It is a very traditional method and started at an early age. As we can see from Figure 7, two of these keys pointing to the same value is called a collision. A good hash function tries to minimize the collision and gives a result that fits in the defined table size. The computing time of the hash function should be minimum with effective value creation. The collision problem has many approaches to resolve such as double hashing, linear probing, quadratic probing [17]. In the proposed project, we have used a signed hash to resolve the collision problem.

A hash function should map keys as evenly as possible so that the probability distribution should be even and would produce uniform results. This is important because if there are many collisions in one place and other values are blank, it would take more time to go through collided values and the other memory part of the system remains unused. Alternately, if the hash function produces results in such a way that it utilizes a complete table to store values and a minimum number of collisions, it would result in less time to go through values and lead to better performance overall. The efficiency of the hash function depends upon its performance and data storing abilities. There is always a trade-off between these two. For larger applications, an ideal hash function should minimize the collision with larger storing space. A hash function also has an important property that it should produce the same results every time for the same key.

2.3.2 HASH2VEC VECTORS

Hash2vec is a technique to produce vectors from the given word using a deterministic approach. It is a numerical representation of words in a low dimensional space vector. The process of producing these vectors would be discussed in detail in the coming sections. The idea behind this methodology is that the traditional method of creating vectors to represent each word in a low dimensional space required a lot of training by applying it to the Neural Network. On the other hand, the hash2vec method does not require any training, it simply tries to get the context from the given context window and tries to produce a hash of the word.

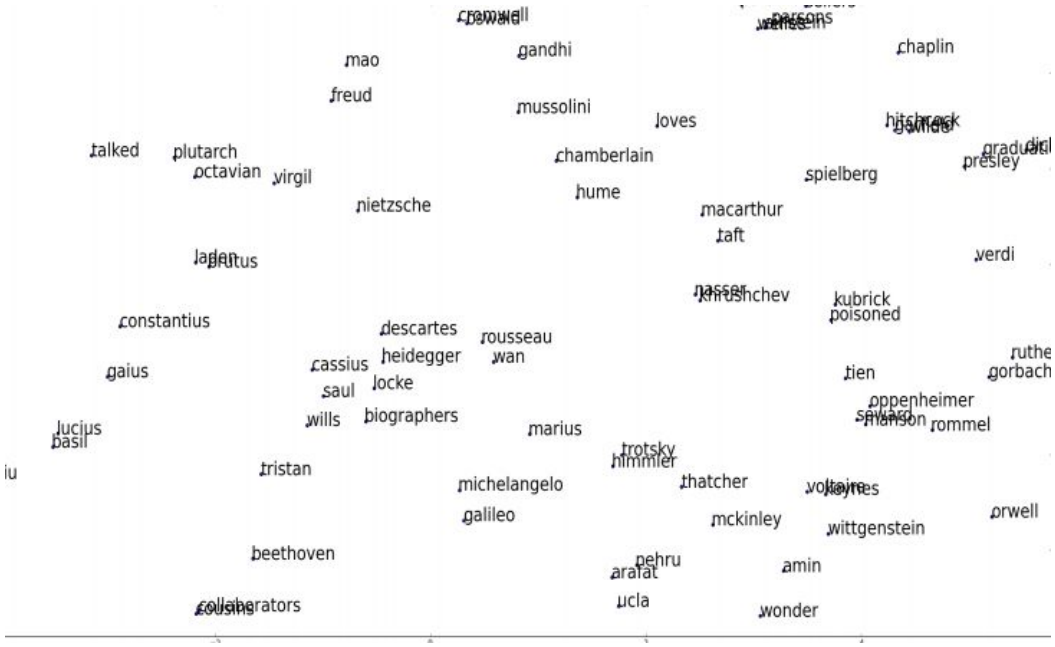


Figure 8. hash2vec vectors represented with their words in the space [1]

The hash2vec method considers the context of the word and processes words with given context length. It stores hash values of the words in the form of a dictionary or hashtable in the temporary storage system. When the word is encountered again in the corpus, it updates the existing hash value of the word. This process is called hashing with context and would require a lesser amount of time as compared to the word2vec training phase. As shown in Figure 8, the words with similar trends such as the group of philosophers, movie directors, famous greek person's names are placed near to each other in the predefined vector space.

2.4 LINEAR REGRESSION

Linear Regression has been used in statistics from so many years. Linear regression is a model that has input variables (X) which are directly related to the one output variable (Y). The input variable has a linear relationship with the output. When there is only one input variable it is called simple linear regression, when there are many input variables it is called multiple linear regression.

Linear regression can be simply explained using the line equation

$Y = m * X + C$

where X is the input value, Y is the output value, m and C are constants. This is the simplest form of linear regression where the X combined with constants can predict the output (Y) value.

In the case of higher dimensions, the equation changes with different numbers of coefficients such as plane or hyperplane. The complexity of this model depends upon the number of coefficients in the equation. If the coefficient from the equation becomes 0 it will make input (X=0) to become zero and there will not be any relevant prediction that exists. Hence the input variable and the coefficient have a direct impact on the model. Linear regression can be applied in many scenarios when there is a linear relationship between input and output.

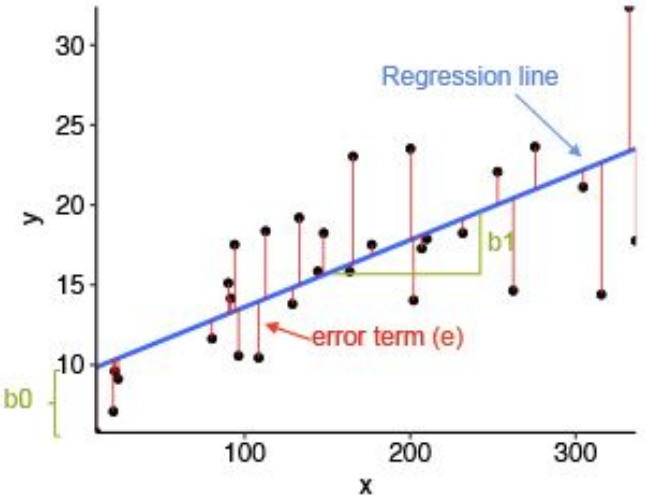


Figure 9. Linear Regression Visualization

One such example is predicting the weight of a person given the height, this can be represented with the equation as given in Figure 9:

$$w = b_0 + b_1 * h$$

In this case, the dataset should contain certain height and weight data of many individuals. It would help in building the model and predicting the weight of a person given his height for the unseen data in the future.

There are different approaches to evaluate the coefficients if there is more than one input value present. One approach is called Ordinary Least Squares [9] where the regression line is drawn through data and the distance between the line and data point is calculated, then all the available distances are squared and added together to get the total sum of errors for each point from the line. This approach tries to minimize this error addition. The second approach is called Gradient Descent [10] where the coefficients are randomly initialized and tried on different input-output data points. Then these coefficients are adjusted in such a manner that the error would be minimized. This process is iterative and goes on until the mean square error reaches the threshold value. This method is preferred when the data set is huge. There are commonly used regularization techniques known as L1 regression and L2 regression.

3. DESIGN AND IMPLEMENTATION

This section focuses on the tools, environment, technologies, and datasets used in this project. We will also discuss data preprocessing strategies and results obtained from the experiments in the further sub-sections.

3.1 ENVIRONMENT DETAILS

Programming Language: Python (Version: 3.7)

Tools and Technologies: NumPy, JSON, bz2, scipy, sklearn, Keras, TensorFlow, word2word dictionary, gensim, google API for word translation, bilingual dictionary.

We preferred Python as a programming language because it has a rich Machine Learning library set which helps in training different models. Also Python is easy to use for converting data from one format into data in another format. As we have used many data format files in this project, it became easier with the Python framework. We have extracted data from the bz2 file into text and then stored it into a JSON file, which we then used as a NumPy array for further processing in different models. These conversions can be efficiently done in Python. Also, Python can integrate with almost all of the third-party applications that include different rest-APIs, dictionaries, external APIs for text translation, etc. Python has another advantage due to libraries like Keras, Tensorflow, etc. that can be easily integrated using Python.

We used the Tensorflow 2.2.0 which has a built-in Keras library. Google has an excellent provision of a collaboratory notebook on google drive. It can access files on google drive with an authentication facility. We preferred collaboratory because installing any library of the desired version is easy on collaboratory and it has a special functionality that allows us to test the piece of code at each step of the implementation. For instance, if any function is written in the collaboratory and you need to be tested immediately, that can be done on the collaboratory by calling the function on the next line only. It also helps in doing collaborative work on the same project. Google collaboratory provides a free GPU for limited hours every day. This has helped us in training larger datasets with millions of words. We wanted to automate the English to Hindi dictionary to fetch into the linear regression model. Hence we also utilized google external API for ordering and testing of data for the experimental setup.

3.2 DATASET

This project had many models and each model required different datasets. There were many parallel datasets available for text translation from English to Hindi. Also, many Wikipedia pages are available in both languages. But some Wikipedia pages do not have an exact translation in both the languages for respective pages. The project demanded the dataset to be rich in vocabulary by maintaining its term frequency in respective languages. We preferred the dataset where each page has text in English and corresponding translated text in Hindi. There were many other options available, such as crawling the websites and collecting the data but due

to the availability of different datasets in both languages we preferred to go for Wikipedia datasets.

In this project, we used different datasets for different models. For simple word2vec vector generations, we used nltk's in-built corpus which has over 3 million words. We also tried to generate vectors with English Wikipedia dumps and experimented with 1 million words. For RNN with sequence to sequence model using the encoder-decoder approach, we used a parallel corpus from Anki [23]. It contains sentences in English and Hindi with a tab-separated structure. We used the same dataset for text translation using hash2vec in sequence to sequence model. For generating vectors from hash2vec methodology, we used Wikipedia corpus in English and Hindi language. We collected wiki dumps of size 1GB with over one million words in each language. Then we applied some preprocessing techniques and used them in the respective models of the project.

3.3 FLOW CHART

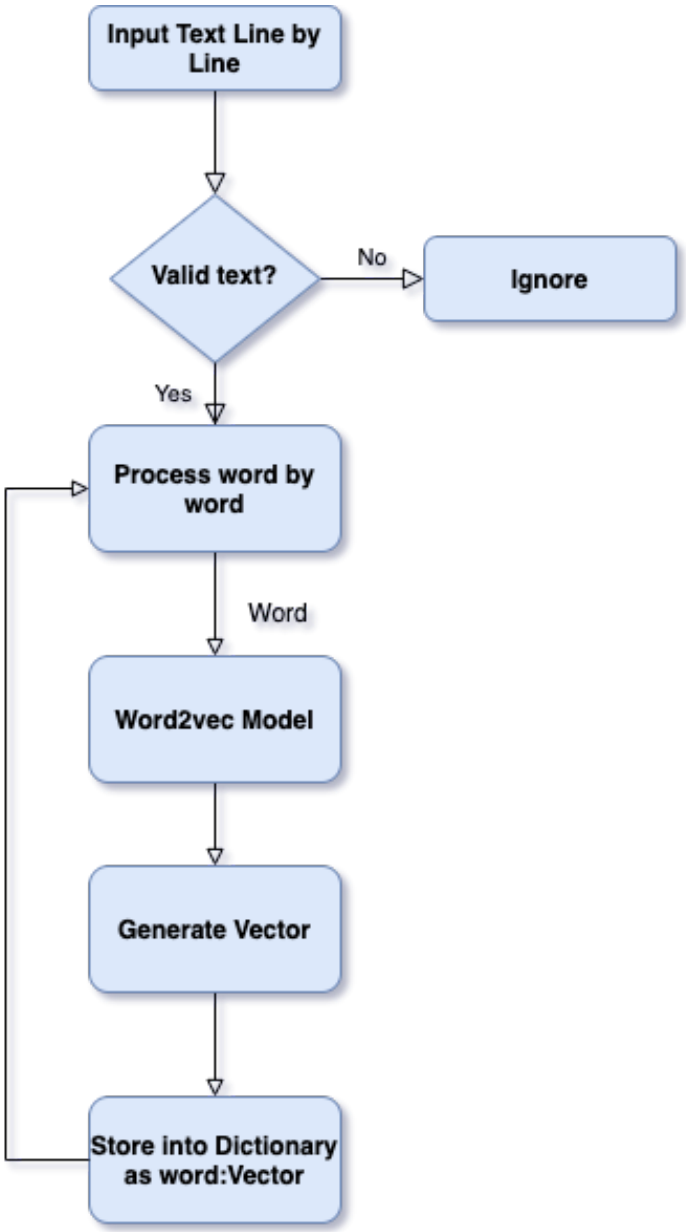


Figure 10. Workflow for word2vec model

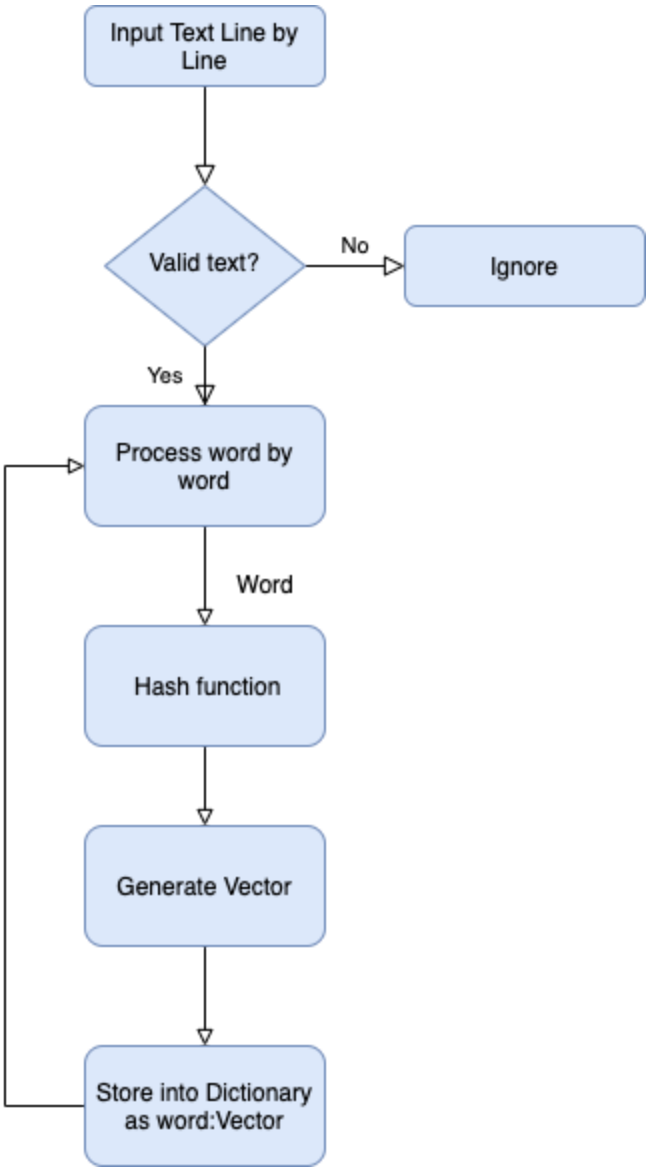


Figure 11. Workflow for hash2vec model

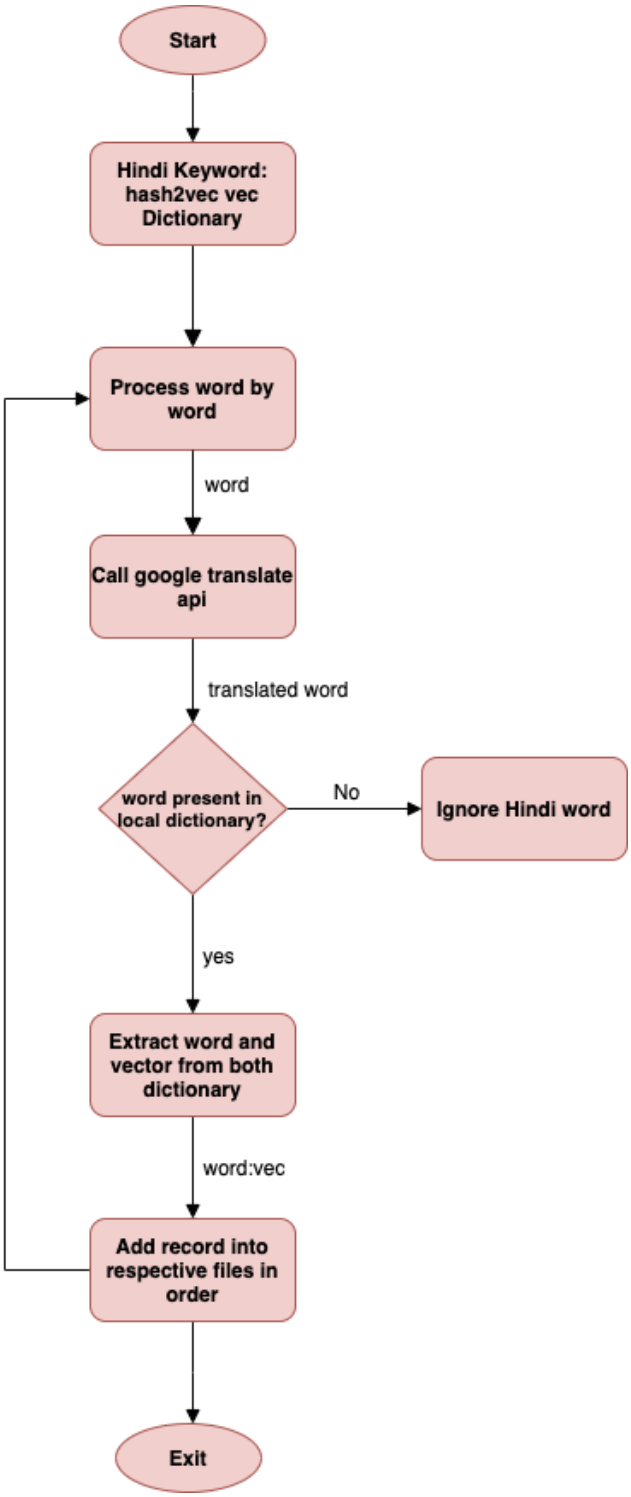


Figure 12. Design flow of automated dataset creation for experimentation

Figure 10 shows the vector generation with the word2vec model using sequence to sequence translation employed with RNN. This model utilized Tensorflow and Keras built-in functionalities in the training phase. Figure 11 shows the workflow for hash2vec vector creation. This unit demonstrated vector creation by applying a hash function to the words in the corpus. This hash2vec model required some data preprocessing activities which we will discuss in the next section. Figure 12 shows the flow of the data ordering using google external API. The data ordering process was a major step in preparing the dataset for the experimental setup (a linear regression model). This automation using google API helped in preparing the word to vector dictionary for both English and Hindi. This automation process was used after both word2vec and hash2vec models generated their respective vectors.

3.4 DATA PREPROCESSING

We followed below steps to do data eneration:

1) Download and extract the dataset or use an online dataset by giving its URL.
2) Remove special characters: replace special characters by a space character.
3) Split words by space.
4) Tokenize the word.
5) Create the dictionary for word to vector.

For the RNN model, there were slight changes in the data pre-processing, below are the steps we used for data preprocessing in both the word2vec and the hash2vec models using RNN:

1) Download the parallel dataset.
2) Split the dataset by tabs.
3) Remove special characters.
4) Add <start> and <end> token to the sentences.
5) Create a dictionary of word to id and a reverse dictionary for indexing.
6) Create a map of word_id to vector to differentiate the vectors by terms.
7) Pad the statement to the defined length.

Sample sentence:

source language

```
1 ----> <start> 29 ----> she 195 ----> wants 9 ----> to 109 ----> work
34 ----> at 7 ----> the 380 ----> hospital 3 ---->. 2 ----> <end>
```

Target language

```
1 ----> <start> 121 ----> तिला 27 ----> त्या 616 ----> रुग्णालयात
53 ----> काम 259 ----> करायचं 4 ----> आहे 3 ----> . 2 ----> <end>
```

3.5 PROJECT ARCHITECTURE

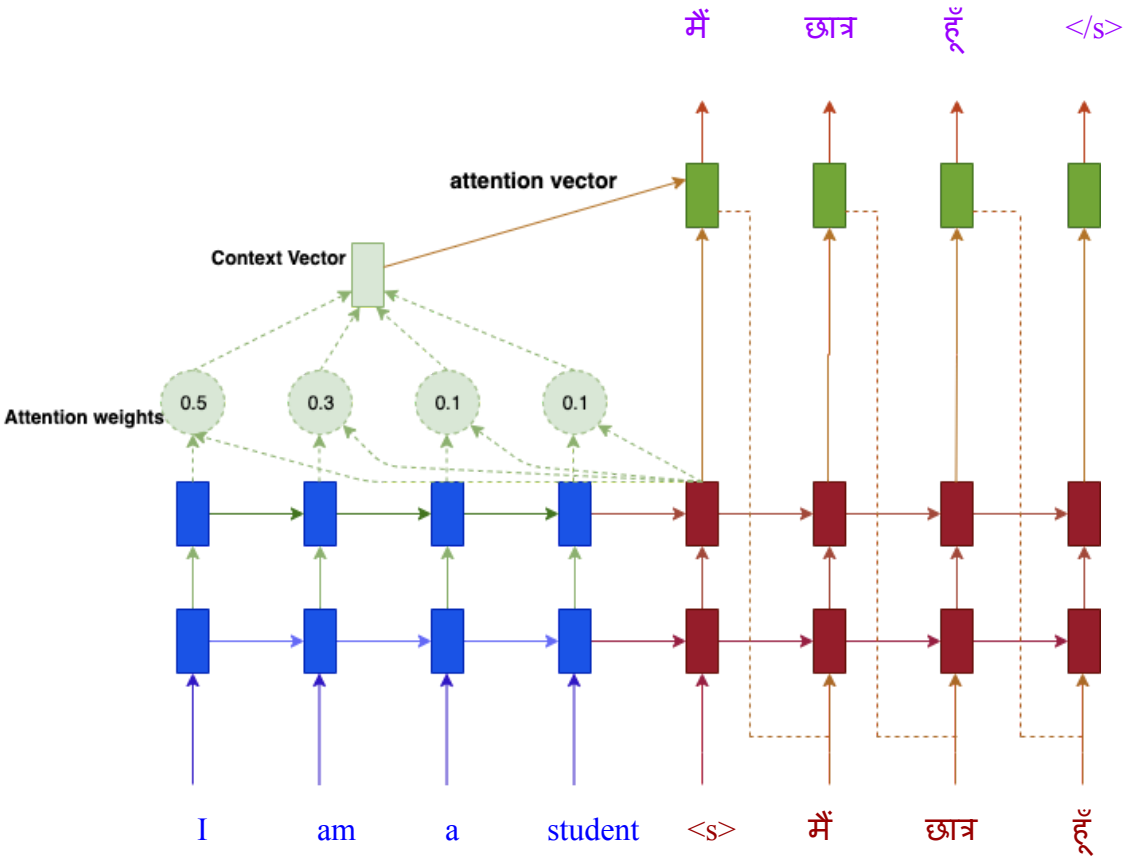


Figure 13. Project Architecture

Figure 13 depicts the overall architecture of the RNN encoder-decoder model used for the project. The RNN model has different layers in its architecture, at the embedding layer the word index gets updated with the respective vectors. In this project, we used the vectors from word2vec and hash2vec techniques as input to the embedding layer. To overcome the overfitting of the information, we added dropout layers to the neural system. These layers arbitrarily drop the hubs in the system at every cycle with the indicated factor. We utilized a softmax function.

The softmax function gives the likelihood for every conceivable word at each timestep from which the most extreme likelihood is picked. We integrated Adam Optimizer in the network to calculate the gradient and boost the overall performance. We used 1024 hidden layers and a batch size of 64. The generated vectors were of 254 dimensions. We tried to use 100,000-word pairs with 100 epochs. The training took around 18 hours to complete with google GPU.

3.6 RESULTS

Below we present some examples of the outputs computed when the word2vec and hash2vec models were used. Output1 represents the result from the word2vec method while Output2 shows the result from hash2vec.

Test Result 1:

Input Text: This is not funny
Expected output: यह मजाक नही है
Output1: यह मजाक नही है
Output2: यह मजाकिया नहीं

Here the word2vec model seems to be working fine with all the words in the output whereas the hash2vec model seems to be partially correct.

Test Result 2:

Input Text: It is not a rocket science
Expected output: यह कोई रॉकेट साइंस नहीं है
Output1: यह कोई रॉकेट साइंस नहीं है
Output2: यह कोई रॉकेट साइंस नहीं

Here both models seem to be working fine with the expected output.

Test Result3:

Input Text: I appreciate your efforts
Expected output: मैं आपके प्रयासों की तारीफ़ करता हूँ
Output1: मैं आपके इफरात की सराहना करता हूं
Output2: मुझे आपका प्राण पसंद

Here the word2vec model seems to be working partially whereas the hash2vec model gives irrelevant results.

Test Result4:
Input Text: Here
Expected output: यहाँ
Output1: यहाँ
Output2: यहाँ

This is experimental results for word2vec and hash2vec using linear regression. Both of the models are giving correct output for the most commonly used word.

Test Result 5:
Input Text: Appreciate
Expected output:तारीफ़
Output1: तारीफ़
Output2: सराहना

Here, the word2vec model produced the correct result whereas hash2vec failed to do so.

Test Result 6:
Input Text: memorize
Expected output: याद
Output1: मेम
Output2: मुझे

In this case, both the models produced the wrong results.

For testing the performance, there were multiple parameters to be considered. First, the time taken to train the model for generating vectors in word2vec and hash2vec models. Second, the translation accuracy of the text produced by sequence to sequence models and linear regression models. In this project, we used human translators to find the accuracy of the sentence translation using RNN word2vec and RNN hash2vec models. For the models with linear regression methodology (experimental word2vec and experimental hash2vec), we employed google external API as a translator to measure the accuracy of the word translation.

System Model	**Number of relevant sentences**	**Number of irrelevant sentences**
word2vec	85%	15%
hash2vec	60%	40%
Experimental Word2vec	75%	25%
Experimental hash2vec	55%	45%

Table 1. Results of the project in the percentage

BLEU Scores:

BLEU or the Bilingual Evaluation Understudy is widely used in the field of Machine Translation. This technique is famous because it works similar to the human brain while scoring the language-translation models. This scoring technique plays an important role in automating the testing for text translation. As the human tries to judge the similarity with his/her own

translated sentence with the machine-translated sentence, the BLEU technique works exactly in the same manner. It reduces the human dependency for calculating the accuracy. Let's discuss with an example:

Expected sentence: This is an amazing film. ====> 5 words

Output sentence: This is amazing. ====> 3 words from the expected result.

score = ⅗

The BLEU scoring range is from 0 to 1.

The main concern about this scoring is that it is one dimensional in nature, viz. if the output sentence has words similar (in meaning) to the reference sentence, it would not be considered while scoring. So the semantic relationship of words is not considered while having BLEU scoring technique. This is the main drawback of this technique. As this technique is world-wide used and accepted for scoring the text translation, we scored using BLEU and got below results.

System Model	**BLEU score**
word2vec	0.75
hash2vec	0.5

Table 2. BLEU scoring on seq to seq model with stops words

Table2 shows the results from the BLEU scoring method applied to sequence to sequence model. The scoring considers stop words like “The”, ”a”,” an”, etc.

System Model	BLEU score
word2vec	0.8
hash2vec	0.65

Table 3. BLEU scoring on seq to seq model without stops words

Table 3, shows how the accuracy has improved when scored without stop words. Since BLEU considers the position of the word in a sentence when scored without stop words, it omits the stop words and hence the accuracy seems to be improved.

Language detector scores using the PCA model:

We found an interesting result for the model with a linear regression algorithm (experimental word2vec and experimental hash2vec). Though the experimental hash2vec model does not seem to be very fluent in the translation of bilingual words, it seems to be working fine for language detection. We performed a small experiment, using the Principal Component Analysis (PCA) model. Word2vec and hash2vec vectors were administered to this model. The PCA model was scored using the log-likelihood scoring method. The input vector belongs to English space if the log-likelihood score of English space is more and similar in the case of Hindi space.

System Model	% correctly detected words
10000	40%
20000	50%
50000	70%
70000	88%
100000	90.3%

Table 4. Results from the PCA model for language detection using word2vec

System Model	% correctly detected words
10000	28.5%
20000	40.4%
50000	70%
70000	78.4%
100000	81%

Table 5. Results from the PCA model for language detection using hash2vec

From Table 4 and Table 5, we can see, as the dataset size increases the accuracy of the language detection model increases. We tried to automate this process by creating a labeled test dataset and running it through the language detection model.

4. CONCLUSION AND FUTURE WORK

The main goal of this project was to compare the time complexity and translation accuracy of word2vec with the hash2vec model when applied to Machine Translation. Different approaches for implementing word2vec models were researched and we followed with the basic Skip-gram approach which has shown superior results in the past [3]. The hash2vec model is not commonly used to solve the Machine Translation problems.

The word2vec model could perform an effective sentence to sentence translation as compared to the hash2vec model. The results from this project show that the translation accuracy of the word2vec model is greater than the hash2vec model. However, the training time required to train the word2vec model was empirically three times more than the processing time of the hash2vec model. Both word2vec and hash2vec models performed almost equally for the language detection and finding similarity of the word within the same language applications. While hash2vec limits the generalizability of the results, it provides new insights into the fact that the vectors can be generated using a deterministic approach.

This project can be enhanced with the bigger dataset and changing the number of epochs used while performing sequence to sequence translation using an encoder-decoder model. Further research can be carried out using the hash2vec model by applying different hashing techniques and minimizing the collision effect. Additionally, the hash2vec model can be further explored to solve problems within the same language.

REFERENCES

[1] Luis Argerich, Matias J. Cano, and Joaquin Torre Zaffaroni: Hash2Vec: Feature Hashing for Word Embeddings(2016).

[2] Hill, F., Cho, K., Korhonen, A., Bengio, Y.: Learning to understand phrases by embedding the dictionary. arXiv preprint (2014).

[3] Mikolov, T., Yih, W. T., Zweig, G.: Linguistic Regularities in Continuous Space Word Representations. In HLT-NAACL, 746–751 (2013).

[4] Pennington, J., Socher, R., Manning, C. D. : Glove: Global Vectors for Word Representation. In EMNLP, Vol. 14, 1532–1543 (2014).

[5] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., Dean, J. : Distributed representations of words and phrases and their compositionality In Advances in neural information processing systems,3111–3119 (2013).

[6] Shi, Q., Petterson, J., Dror, G., Langford, J., Strehl, A. L., Smola, A. J., Vishwanathan, S. V. N.: Hash kernels. In International Conference on Artificial Intelligence and Statistics, 496–503 (2009).

[7] Attenberg, J., Weinberger, K., Dasgupta, A., Smola, A., Zinkevich, M.: Collaborative Email-Spam Filtering with the Hashing Trick. In Proceedings of the Sixth Conference on Email and Anti-Spam (2009).

[8] Weinberger, K., Dasgupta, A., Langford, J., Smola, A., Attenberg, J.: Feature hashing for large scale multitask learning. In Proceedings of the 26th *Annual International Conference on Machine Learning*, 1113–1120 (2009).

[9] H. S. Simon and A. Purwarianti, “Experiments on Indonesian-Japanese statistical machine translation,” *Comput. Intell. and Cybernetics* (CYBERNETICSCOM 2013) *IEEE Int. Conf.* 2013, pp. 80-84.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L.Jones, A.N. Gomez, L. Kaiser, I. Polosukhin, "Attention is all you need," A*dvances in Neural Information Processing Systems*, pages 6000–6010, 2017.

[11] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ICLR*, 2015.

[12] C. Monz, "Statistical machine translation with local language models," *Proceed. of the Conf. Empirical Methods in Natural Lang. Processing*, 2011, pp. 869-879.

[13] I. Goodfellow, D. Farley, M. Mirza, A. Courville, and Y.Bengio, "Deep Learning based Machine Translation in Maxout networks," *Int. Conf. Machine Learning IMCL*, 2013, pp. 439-444.

[14] P F. J. Och and H. Ney, "The alignment template approach to statistical machine translation," *Comput. Linguist.* vol. 30 no. 4, 2004, pp. 417-449.

[15] A. Birch, M. Osborne, and P. Koehn. 2007. "CCG supertags in factored statistical machine translation" *Proceed. of the Second Workshop on Statistical Machine Translation,* pp 9–16.

[16] genism, https://radimrehurek.com/gensim/

[17] Microsoft Translate, https://www.microsoft.com/en-us/translator/

[18] Linguee, https://www.linguee.com/

[19] Hash function, https://en.wikipedia.org/wiki/Hash_function

[20] Neural Machine Translation with attention using tensorflow, https://www.tensorflow.org/tutorials/text/nmt_with_attention

[21] Google Colaboratory, https://colab.research.google.com/

[22] A beginner's guide to word2vec, https://skymind.ai/wiki/word2vec.

[23] RNN neuron, https://en.wikipedia.org/wiki/Recurrent_neural_network

[24] Google Cloud Vision, https://cloud.google.com/vision/

[25] Anki dataset link https://www.manythings.org/anki/

[26]T. Brants, A. Popat, P. Xu, F. Och, and J. Dean. “Large language models in machine translation”. *Proceed. of the 2007 Joint Conf. Empirical Methods Natural Lang. Proces. and Comput. Natural Lang. Learning* (EMNLP-CoNLL), pp 858–867