UNIVERSITY of LIMERICK

OLLSCOIL LUIMNIGH

Text Classification/Analysis Using Neural Networks and Word2Vec: How to Use The Technique

Author: John Rey Juele
Student ID: 15167798
Supervisor: Farshad Toosi

Interim Report

B.Sc. Computer Systems

Department of Computer Science and Information Systems

Table of Contents

1. Project Summary 3
2. Introduction 4
   2.1. Context/Motivation 4
   2.2. Technologies 5
   2.3. Objectives 6
3. Research 7
   3.1. Artificial Neural Networks 7
      3.1.1. The Inspiration 7
      3.1.2. The Application 8
   3.2. Word2Vec 13
   3.3. Doc2Vec 16
4. Possible Approach To Project 18
5. Current Progress & Future Plan 20
   4.1. Work Accomplished So Far 20
   4.2. Plan of Future Work 24
References 25

1. Project Summary

For my final year project, I am developing an application that will take a piece of text or a set of texts as input and will then label them with a topic or subject that best describes them. This project makes use of concepts and ideas that are brought out in the fields of machine learning and natural language processing and the objective is to use these to create software that fulfils a need in real life.

During the lifetime of this project I am looking into developing an application that incorporate software models capable of performing classification tasks, of which I am paying close attention to neural networks and the theory and the operations that make them work. I am also including “Word2Vec” (Mikolov, et al., 2013), a relatively recent technique devised by a team of researchers at Google. This technique converts words into numerical representations which can in turn help the computer process and understand language while at the same time keeping the semantics of the words intact. Word2Vec can also be extended to cover not just words but entire documents with the help of “Doc2Vec” (Mikolov and Le, 2014).

As the machine learning algorithms that I am using utilises a supervised style of learning, it is required to obtain a good set of training data (Brownlee, 2016). This can range from news articles, academic papers to non-formal texts sources such as Twitter and other social medias. There is no limit to the sources I can pull from as long as they are appropriately labelled due to the fact that the performance of the classification task greatly depends on the quality of the training data. The pre-processing of the text is also required in order to help the machine efficiently handle the text input we feed into it. This may include stemming the words, removing unnecessary punctuations and other pre-processing techniques on text data.

One of the goals of this FYP is to produce working software and therefore I will also be considering the characteristics and best practices to developing good software. I am approaching this project similar to how a business would approach such a development problem by keeping in mind the functional and non-functional requirements, devising a plan of action and to incorporate design patterns so that the application can remain in line with the “SOLID” design principles to software development.

2. Introduction

2.1. Context/Motivation

As we are living in a digital age, information has become widely distributed and therefore easily accessible. Even though we have not completely moved on from physical books and other paper-back publications, more emphasis has been placed on making information available electronically. 80% of the information available to us, both in physical and electronic form, is in text format (Korde and Mahender, 2012).

Helping computers understand and process human language has been an active area of research in computer science especially in the field of natural language processing (NLP). One of the subareas of this field is in the classification of text. Computers are being used as tools to automate work especially in analysing and processing text with a view of classifying them. With this in mind, text classification is a contributory building block in many NLP applications. Following on from this, the technological environment today has encouraged researchers to place more attention in incorporating and implementing mechanisms of text classification in everyday, practical tasks. We can see this in action with the prevalence of machine learning today regarding web searching, sentiment analysis and data filtering (Aggarwal and Zhai, 2012), as well as in digital assistants such as Apple’s Siri and Amazon’s Alexa.

It has always been an interest of mine to dabble in the field of machine learning and to gain an understanding on how the resources of computers can be manipulated to mimic and simulate human-like intelligence and learning. Communication and language is an integral part of being human as well as of society as a whole. I am curious to discover the nuts and bolts that operate underneath the machine learning technology of today, especially on how they perform human-like tasks such as understanding text.

2.2. Technologies

The below are the main technologies that I will be using for this project. I should mention that the list below is not concrete and is subject to change.

Python

Python is the programming language that I am predominantly using to implement the application. The main reason why I decided to use python is because I did not use python previously and the FYP is an opportunity to learn and utilise the language in a meaningful way. Another reason why I chose python is the many libraries available as well as the documentation and tutorials that are available online.

PyCharm

PyCharm is the IDE that I intend on using for this project. It is one of the IDE's provided by Jet Brains who also have an IDE available for the Java programming language called "IntelliJ". I am very familiar with IntelliJ and the features and settings in IntelliJ are also available in PyCharm and it is because of this that encouraged me to use it.

Python Libraries

- **Gensim**

  Gensim is intended to be used to help create my Word2Vec and Doc2Vec models. According to the Gensim website its purpose is to "realize unsupervised semantic modelling from plain text" which of course encompasses what I would like to achieve in my implementation of the natural language processing side of the project.

- **NumPy**

  NumPy is the library that I will be using to perform the mathematical operations within my project. The library also makes available a more efficient implementation of arrays and matrices which I envisage to be useful in my research and possibly the implementation of my neural networks.

- **Matplotlib**

  This library will come in use when plotting and creating graphs. As it is a possible feature of the application to plot and visualise document relationships, matplotlib can aid in the development of such a feature. It also helps in the understanding of some of the concepts of research especially in visualising the output of the neural networks as well conceptualising the ideas provided by word2vec.

- **Scikit-Learn**

  Scikit-learn is a library that provides already made implementations of many machine learning algorithms. This will come in great use in the creation of my classification models as well as the evaluation of such models. The library also have implementations available for pre-processing data including text data.

2.3. Objectives

The main goal of this project is to develop an application that can read the text provided to it and to label each one an appropriate topic or subject. This goal can be broken down even further to three major milestones or sub-goals.

1. **A training and test dataset must be found and readily available**. I will be using a supervised style of learning for my classification model so therefore it is vital that these dataset must be correctly labelled and is suitable for the task at hand. Upon obtaining the dataset, they should also be pre-processed. This includes but is not limited to stemming the words in the document, removing unnecessary punctuation and “stop words”.

2. **Develop the program using the SOLID design principles.** For the implementation of the program I intend to follow a 3-tiered closed architecture as well as including the MVC (Model View Controller) architectural pattern. It is also a goal

of mine to add in necessary design patterns with a view of accommodating quality attributes such as flexibility and extensibility. The program will of course include a graphical user interface that will display not just the labelling of the documents but also the visualisation and graphing tools that I intend to add to the application. I plan to design the GUI as user friendly as possible by making it easy to use, clear and organised.

3. **Test and evaluate the end product**. Upon finishing the implementation, I will create different test cases for the outputs of the program. I also aim at evaluating the classification model that I will be incorporating into the program and to determine the accuracy of said model. Using the data from the tests I will be trying to tweak the different parameters of the program with a goal of increasing and improving the accuracy of the classification model.

3. Research

3.1. Artificial Neural Networks

An artificial neural network (ANN) is an “information processing paradigm” that takes inspiration from the workings and the operation of the biological brain (Stergiou and Siganos, n.d.). ANN’s are not necessarily an algorithm; that is it follows a specific and concrete set of steps to solve a particular problem. They can be seen more as a paradigm or a framework as to how to structure and conceptualise the program.

3.1.1. The Inspiration

Artificial neural networks are inspired by our brains which are comprised by a large network of neurons. A neuron is just a single nerve cell that transmit electrical signals from one end to the other.

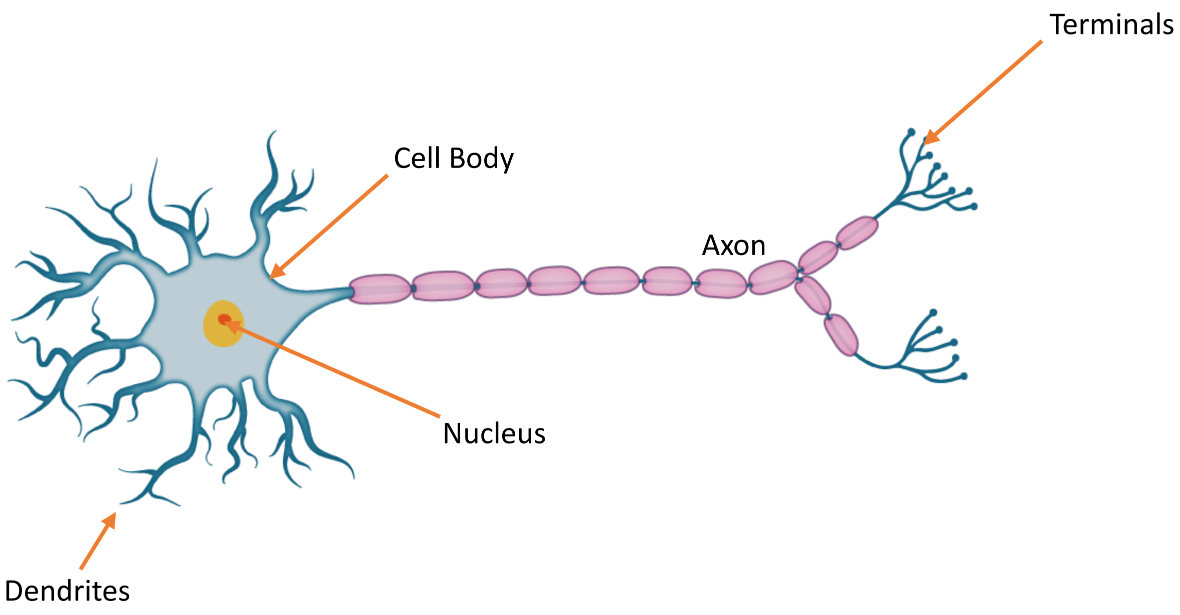


Fig. 1. A Neuron

In the case of the diagram above, the electrical signal would enter the neuron through the dendrites which will then be processed inside the nucleus. It is only upon reaching a certain threshold that an output signal would be produced which is then sent through the axon and exists through the synapses within the neuron's terminals (Rashid, 2016, pp. 41-43).

3.1.2. The Application

We can model an artificial neuron like so:

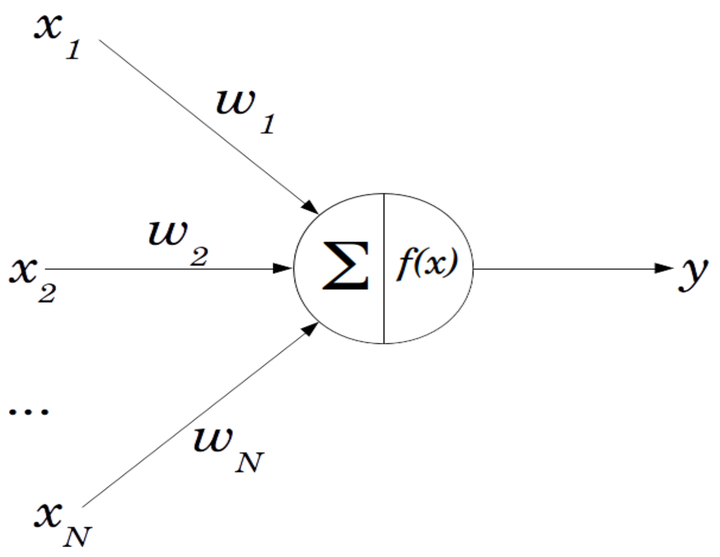


Fig. 2. A Perceptron

The official name of an artificial neuron is a “perceptron” which can act as a single processing unit of an ANN (Rosenblatt, 1958). The main components of the perceptron are:

- **The weighted inputs**

  There can be N inputs to this neuron which are also weighted. The weights on each connection signifies the strength of that connection and therefore influences the processing and ultimately the output of that perceptron. These weights are assigned random values initially. They can be seen as the dendrites from the biological neuron.

- **The neuron**

  Within the neuron itself is where the processing takes place. Two main processes take place. The “∑” or sigma signifies the summation of the inputs in the form:

$$sum = (x_1 * \omega_1) + (x_2 * \omega_2) + \cdots + (x_N * \omega_N)$$

  As stated earlier under the biological brain, a certain threshold has to be met before the output is fired out of the neuron. This is modelled using an activation function: $f(x)$ where we pass the sum into that function. There are many types of activation functions such as the step function and the sigmoid/logistic functions (Rashid, 2016, pp. 44-45). The result of that function will be that neuron’s output or in other words its *activation value.*

  Just like the brain is a network of neurons, we can model a network of perceptrons like so:

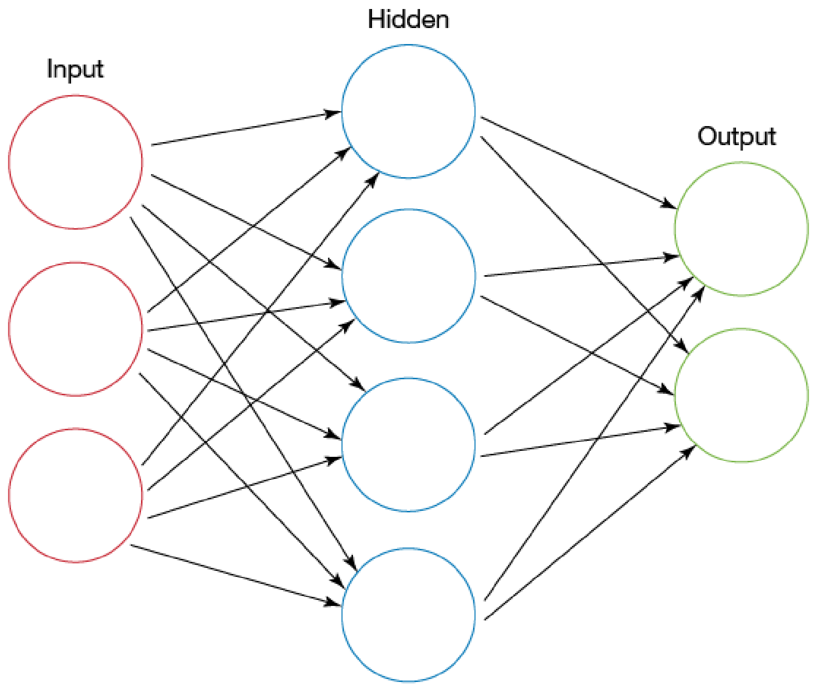
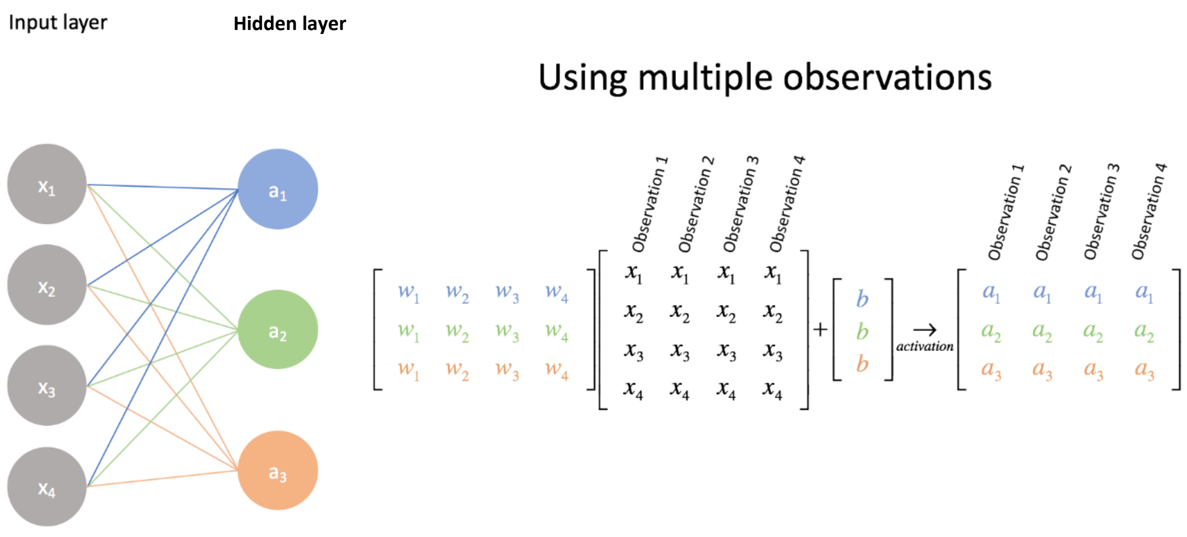


Fig. 3. A feed forward neural network

Each circle is a neuron/perceptron where the output of that neuron will act as the input of the next neuron. I should also note that connections between neurons are weighted (as weights are not included on the above diagram). In a typical neural network architecture, there are three layers: the input, hidden and output layers. The number of nodes in the output and input layers are dependent on the problem domain while the number of nodes in the hidden layer is arbitrary.

In order to produce a guess out of the neural network, it must perform a feed forward mechanism through the network. Once executed, the activation value of each node in each layer is calculated until processing reaches the output nodes. The activation value of the nodes of the output layer is the neural network's guess. I should also point out that there is a more efficient way of performing the feed forward algorithm. Everything on the neural network can be represented as matrices (Nielsen, 2018).



$$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} x_1 & x_1 & x_1 & x_1 \\ x_2 & x_2 & x_2 & x_2 \\ x_3 & x_3 & x_3 & x_3 \\ x_4 & x_4 & x_4 & x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} \xrightarrow{\text{activation}} \begin{bmatrix} a_1 & a_1 & a_1 & a_1 \\ a_2 & a_2 & a_2 & a_2 \\ a_3 & a_3 & a_3 & a_3 \end{bmatrix}$$

Fig.4. Matrix representation of the NN

In order to find the activation values of the nodes in the next layer, a dot product operation can be done between one matrix with another. With the above diagram, we perform the operation between the matrix representing the weights between the input and hidden layer and the matrix representing the input layer in order to produce the activation values in the hidden layer. The diagram above has not included the activation function however. So upon obtaining each values from performing the multiplication, we apply the activation function on each of the values in the result in order to find activation value of the nodes. We can do the same operation in order to find the output layer values. Through linear algebra operations we can simplify the feedforward algorithm especially programmatically.

However running the feed forward on a neural network for the very first time may not operate and produce a result that is expected. That is due to the fact that it is untrained. The neural network must learn from already existing data in order to help it produce a more accurate guess. As I have pointed earlier, the weights between each layer greatly influence the activation values of the next neuron. We can also see them as the decision makers of the network. As the weights are initialised with random values, they will likely produce inaccurate activation values in the next node. So from that idea, when training the neural network, what really is happening in the background is that the weights are being adjusted towards a more accurate value.

A concept found in supervised learning is the idea of errors. This is very similar to how humans learn from making mistakes. We compare our work with the actual answer and to use the mistakes we have learnt to adjust our current knowledge. So in the case of neural networks, we make it produce a guess for a particular set of inputs and to compare its guess with the actual answer that is correlated to that set of inputs. Let’s put this mathematically:

$$error = answer - guess$$

Upon discovering the error of the output layer (as the error produced from the overall guess of the NN is the error of the output layer), we use those errors to traverse or propagate back towards the input layer of the NN while at the same time finding the errors of each node in each layer. This is where the mechanism of “backpropagation” comes in (Klein, 2011).

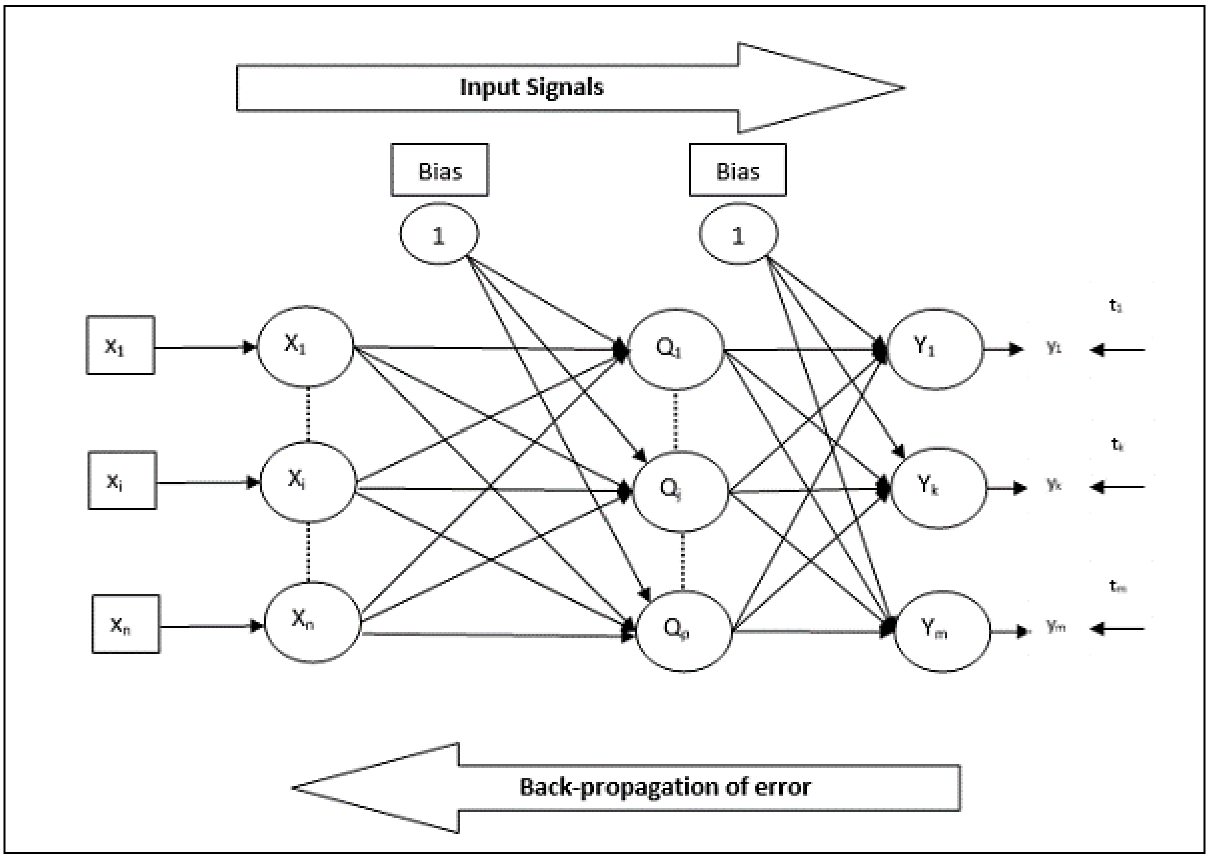


Fig. 5. Feed Forward & Backpropagation

Once we have obtained all the errors of each node in the neural network it will help us to figure out how much we should adjust each of the weights in the networks. The goal of the learning process is the minimize the errors and get them as close to zero as possible. This brings up a new idea of “gradient descent” (Rashid, 2016, pp. 83-93). From this discussion we can conclude that the effectiveness of a neural network is not determined fully by the efficiency of the implementation but by the quality of the training data. As this is a supervised model, the guesses produced by neural networks depend fully on the data it learnt from.

Gaining an understanding on the functionality of a neural network is very important for this project as it is the foundation of many of the core features of the project application. We can see one in use in the next section with regards to Word2Vec.

3.2. Word2Vec

Word2Vec is a mechanism that converts words into “word embeddings” or numerical representations of words (TensorFlow, 2018). There are many different ways of creating word embeddings such as TF-IDF vectorisation, count vectors and co-occurrence matrices. However, what is great about Word2Vec is that it maintains the semantical relationships that exist between words. We can illustrate this by performing matrix calculations between word vectors. For example, the word vector produced by:

$$vector(\text{"King"}) - vector("Man") + vector("Woman")$$

is a vector that is very similar to the word vector of “Queen” (Mikolov, et al., 2013).

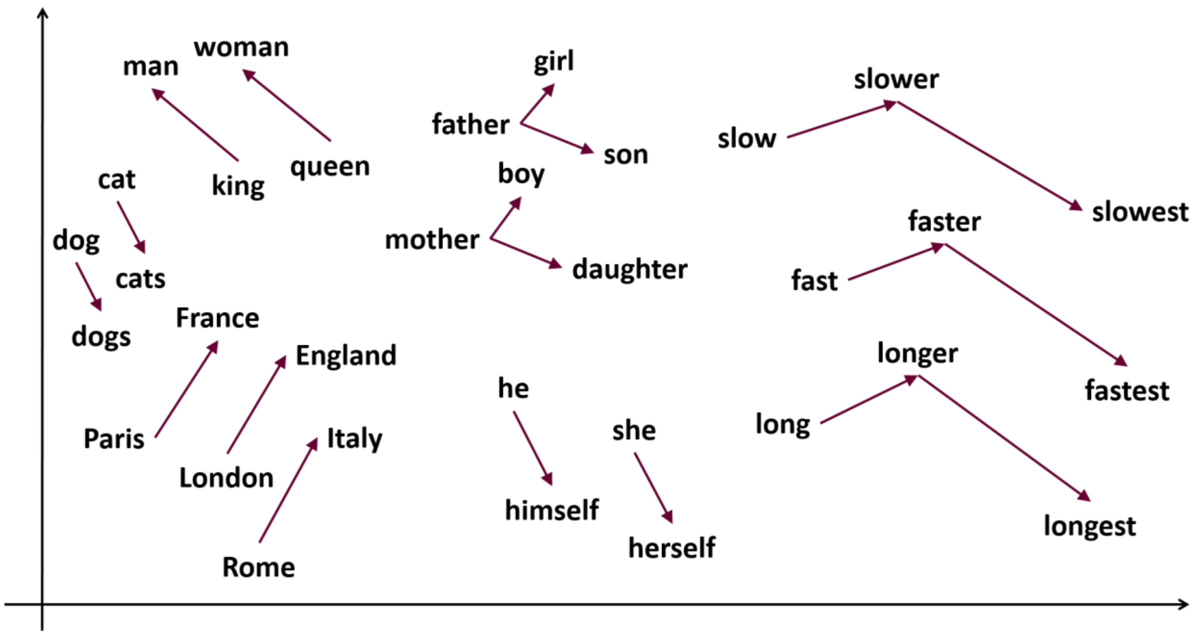


Fig. 6. Vector relationships between words

Word2Vec makes use of shallow neural networks (NN with only one hidden layer) that are trained to do a specific purpose. The two main purposes are described by the two architectures of Word2Vec: **Continuous Bag of Words** and **The Skip-gram model**.

The Skip-gram model

The skip-gram model is a neural network that is trained to predict the neighbouring word (context words) given a specific word in the text corpus.

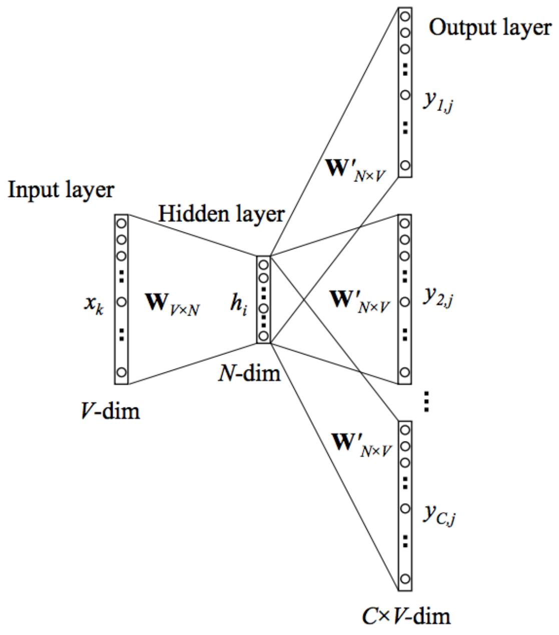


Fig. 7. The skip-gram architecture

Continuous Bag of Words (CBOW)

Continuous Bag of Words is a neural network that is trained to guess a particular word given the word(s) that surrounds that word.

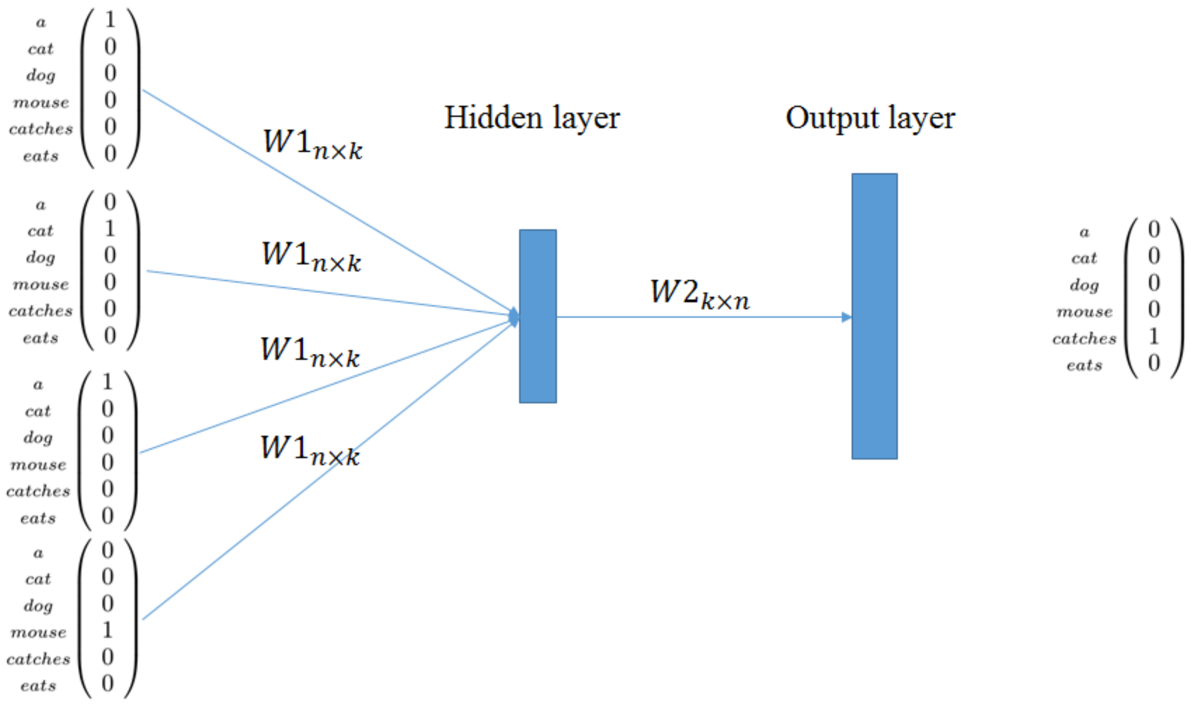


Fig. 8. The CBOW architecture

As these neural networks are being trained to fulfil their target functions, the weights connecting each layer in their networks are being adjusted towards a more accurate value.

It is from these weights that we can derive the word vectors for the words in the text corpus (McCormick, 2016). As I have pointed earlier everything on the neural network can be represented as matrices.

$$\begin{bmatrix} 1.1 & -1.1 & 5.5 \\ 6.3 & 0.02 & 4.2 \\ 2.3 & -1.6 & 4.2 \\ 3.2 & 7.2 & 0.6 \\ 8.3 & 4.9 & 4.1 \end{bmatrix}$$

Fig. 9. The weight matrix representing the weights between the input and hidden layer of a hypothetical Word2Vec model.

We can also represent the input word to the NN like so:

$$[0\ 0\ 1\ 0\ 0]$$

Fig. 10. A one-hot representation of the input word “Computer”

In order to find the word vector for the word “Computer” (As an example) we perform a simple dot product between the two matrices.

$$[0\ 0\ 1\ 0\ 0] * \begin{bmatrix} 1.1 & -1.1 & 5.5 \\ 6.3 & 0.02 & 4.2 \\ 2.3 & -1.6 & 4.2 \\ 3.2 & 7.6 & 0.6 \\ 8.3 & 4.9 & 4.1 \end{bmatrix} = [2.3 \quad -1.6 \quad 4.2]$$

By doing such an operation, what we’re really doing is that we’re selecting the word vector from all the stored weights representing all the possible words in the corpus and this selected word is projected into the hidden layer. So from the above example the word embedding for the word computer is: $[2.3 \quad -1.6 \quad 4.2]$

3.3. Doc2Vec

Doc2vec (Paragraph Vector is what it is called in Mikolov and Le's paper) is the technique used to create embeddings of entire documents. Similar to how word2vec preserves the semantical relationships between words while creating the embeddings, doc2vec maintains similarities between documents that may be related in topic or subject matter (Mikolov and Le, 2014). According to the paper, the texts in question can be of varying length, from sentences to large documents.

Doc2vec is simply just an extension to the word2vec technique. The only difference it has with word2vec is that it involves a paragraph ID or document ID (Mikolov and Le, 2014). This ID has to be unique for all documents involved. Just like word2vec, it makes use of two main models: **Distributed Memory** and **Distributed Bag of Words**.

Distributed Memory Model (PV-DM)

This model can be seen as an extension to the "Continuous bag of words" model from word2vec which can be seen from the below diagram.

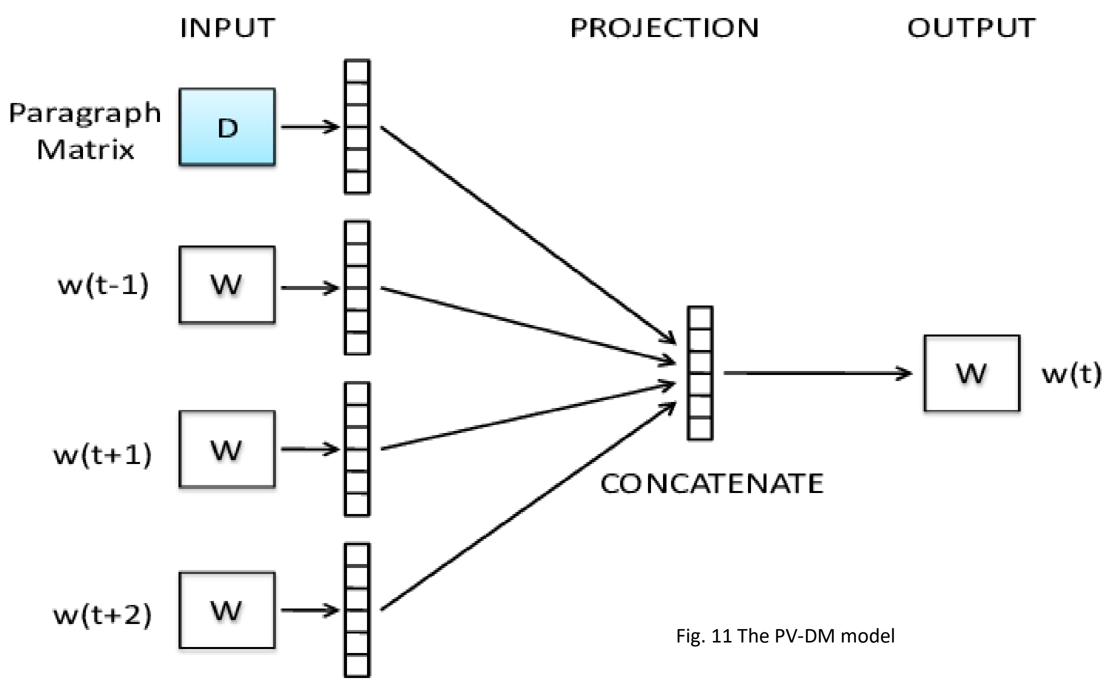


Fig. 11 The PV-DM model

Along with the context words, the paragraph ID is involved as input into the neural network. The neural network is trained the exact same way as CBOW is supposed to be trained but in this case, the document ID is trained alongside the words that belong to that document ID. In order to find the document vector, we use the same method I mentioned in word2vec but instead we use the weight matrix of the documents rather than the words.

Distributed Bag of Words Model (PV-DBOW)

This is the doc2vec variant of the "Skip-gram" model of word2vec. The difference here however is that the document ID is the input to the neural network. This neural network is trained to guess words that belong to that inputted document ID. The advantage to this model compared to the previous one is that it has a lesser memory requirement as we only need to store the document weights and can ignore weights for the words. I should note that in the paper, Mikolov and Le combines both models during their experiments of which they recommend us to do the same. However they state that PV-DM alone is enough to perform most tasks (Mikolov and Le, 2014).

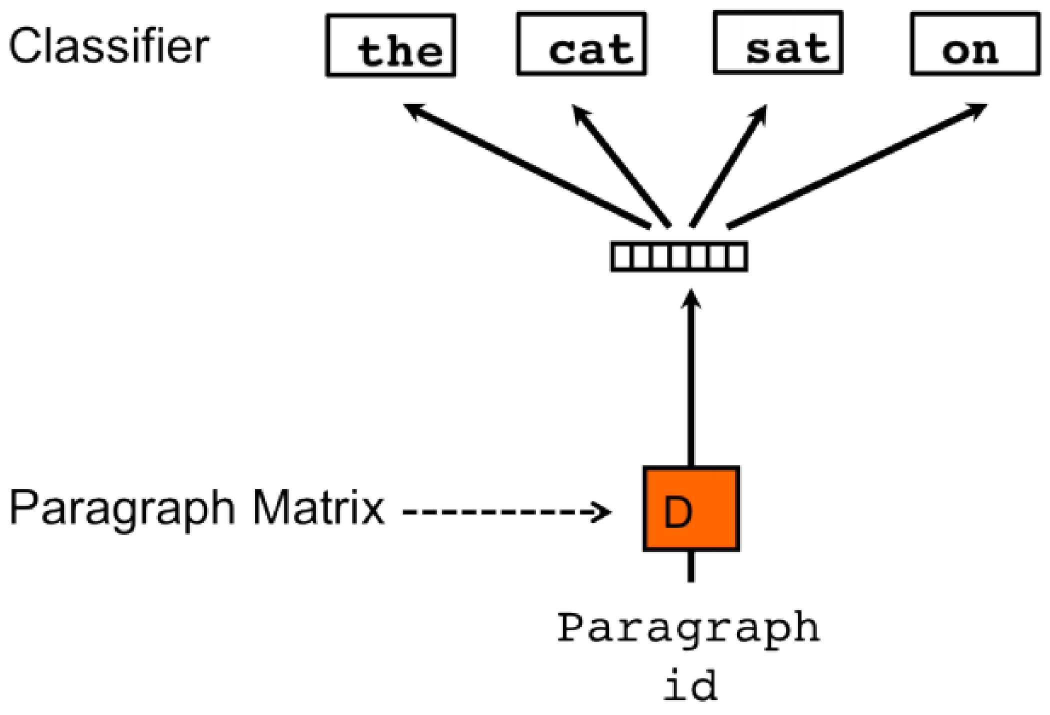


Fig. 12 The PV-DBOW model

4. Possible Approach To Project

As you have probably noticed during the previous sections that each of the research material follows on from each other. In order to makes use of one area you must have an understanding of its foundations. The research I have done will prove to be the building blocks of the final product.

I have thought of some ideas as to how to produce the software application using the research material. In its very essence the software system would make use of two neural networks; one network to produce document embeddings of the input documents using doc2vec and a second network to perform the actual classification using the document embeddings.

Let me explain my possible approach to this a little deeper. The first neural network will act almost like an embedding factory where it is trained using a doc2vec model, either using the distributed memory model or the distributed bag of words model. Once this neural network is trained efficiently, it will then be capable of producing document embeddings, even documents that it has not encountered before. The second neural network will be the classification model and will perform the labelling of the documents. The document embeddings will act as the input to that neural network and the appropriate topic is the output. This neural network is trained with documents that have known topics to them. I plan to encode the topics as one-hot vectors as well so that it makes it easier to add more if needs be. I should also note that I am not limited to using a neural network as the classification model. I can very much use other models like Naïve Bayes, Support Vector Machines, K-Nearest Neighbour and others to perform the labelling of the documents. I will look into these classification algorithms deeper as I progress.

As I have stated in previous sections of the report, the use of a supervised model (with neural networks or other supervised classification models) calls for the availability of training and test datasets (Brownlee, 2016). The role of the training dataset is to adjust and enhance the predictive parameters of the classification model. Currently I have access to the ACM, IEEE and many other online sources of scientific papers as I am a current student of the university. I also have in mind news articles that are available online which are great for automatically being tagged to topics and subjects. It is possible for me to create a web scraper in python to automatically pull the main text from the articles as well as the tags and topics associated with the article. I have also been given a text dataset of news articles from BBC which will be a good addition to the dataset collection. Another source I have in my disposal is Kaggle, a popular website among data scientists where a large repository of different types of datasets are available. Finally I should state that I have access to the Twitter API. With this API, I can search for tweets by hashtags as the hashtags will act as the topic of that tweet. These datasets, after being pre-processed and clean will help train the classification model as well as help test the accuracy and the performance of the model.

Another feature that I could add to the application is to add a visualisation of the relationships between documents similar to the diagram included below:

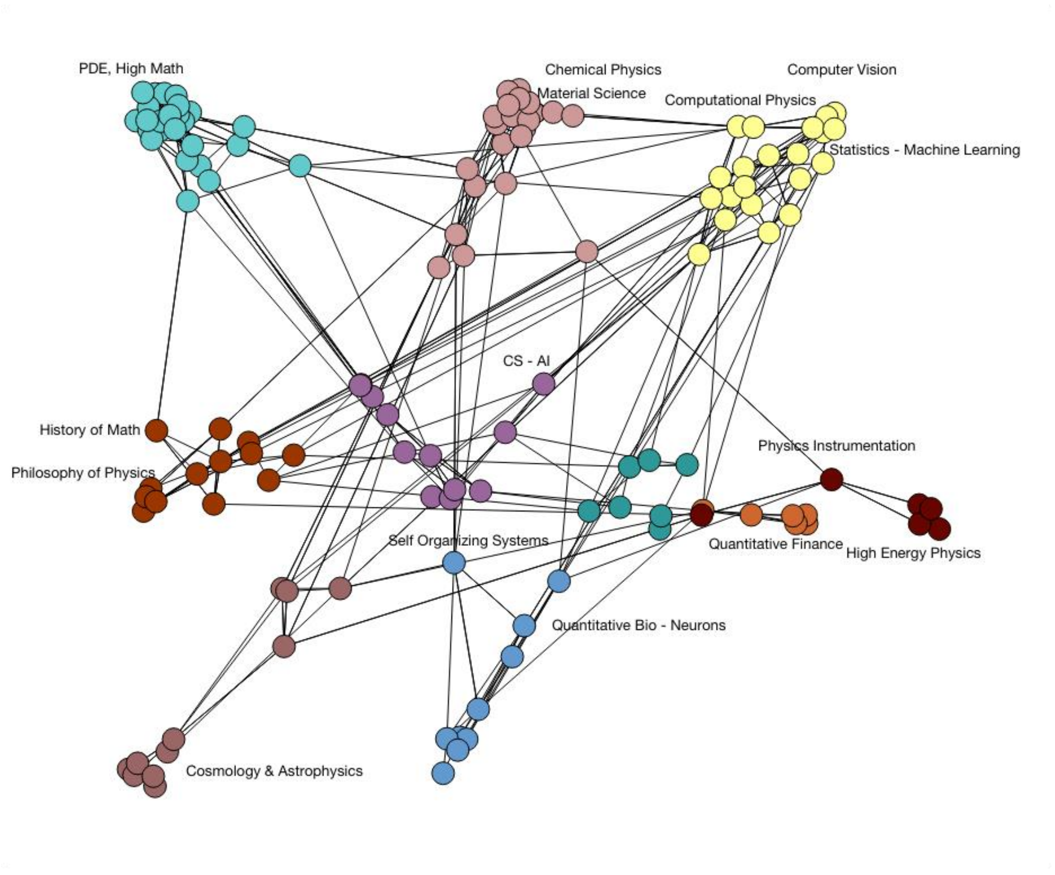


Fig. 13: A possible graph I can include to the program that visualises document relationships.

The user can input multiple documents into the program and it would create a graph or diagram to visualise the relationships between each one. I hope to also display the strength of the relationships and to showcase which documents are similar to each other by topic.

Due to the nature of doc2vec and word2vec in which they preserve the meaning and semantics, this feature is by no means impossible to implement and is something I can possibly include to the end product.

As I have not fully started in the implementation, these ideas are not guaranteed to work as I have explained above. This is just an hypothetical plan which of course I will be following but it is possible for it to change as I advance deeper into the project.

5. Current Progress & Future Plan

4.1. Work Accomplished So Far

I spent the majority of the 1st semester doing research on all the necessary components of the project. I wanted to deepen my understanding on the concepts I mentioned earlier in order to comprehend the terms used in the libraries that I will be using as well as to have an idea to what is actually happening under the hood. I did so by creating mini programs of the concepts I explored in order to be acquainted with the operations that takes place to make it work.

Below is the class I coded for my simple **neural network**:

The Class constructor:

```
class NeuralNetwork:

    def __init__(self, i, h, o, learning_rate):

        # set the number of nodes for each layer
        self.input_nodes = i
        self.hidden_nodes = h
        self.output_nodes = o

        # weights
        self.weights_to_hidden = np.random.rand(self.hidden_nodes, self.input_nodes)
        self.weights_to_output = np.random.rand(self.output_nodes, self.hidden_nodes)

        # biases
        self.biases_hidden = np.random.rand(self.hidden_nodes, 1)
        self.biases_output = np.random.rand(self.output_nodes, 1)

        # learning rate
        self.learning_rate = learning_rate
```

The feed forward function:

```
def feed_forward(self, inputs):

        # Convert input to column vector
        inputs = np.array(inputs, ndmin=2).T

        # Compute hidden node activations
        hidden_values = np.dot(self.weights_to_hidden, inputs)
        hidden_values = np.add(hidden_values, self.biases_hidden)
        hidden_values = activation_function(hidden_values)

        #Computer output node activations
        output_values = np.dot(self.weights_to_output, hidden_values)
        output_values = np.add(output_values, self.biases_output)
        output_values = activation_function(output_values)

        return output_values
```

The training function:

```
def train(self, inputs, answers):

        # Convert parameters to column vectors
        inputs = np.array(inputs, ndmin=2).T
        answers = np.array(answers, ndmin=2).T

        # Get a vector of all the guesses the neural network makes
        hidden_values = np.dot(self.weights_to_hidden, inputs)
        hidden_values = np.add(hidden_values, self.biases_hidden)
        hidden_values = activation_function(hidden_values)

        output_values = np.dot(self.weights_to_output, hidden_values)
        output_values = np.add(output_values, self.biases_output)
        output_values = activation_function(output_values)

        # Adjust weights hidden -> output
        output_errors = np.subtract(answers, output_values)
        temp = output_errors * output_values * (1.0 - output_values)
        temp = self.learning_rate * temp

        delta_weights_ho = np.dot(temp, output_values.T)
        self.weights_to_output = np.add(self.weights_to_output, delta_weights_ho)
        self.biases_output = np.add(self.biases_output, temp)

        # Adjust weights input -> hidden
        hidden_errors = np.dot(self.weights_to_output.T, output_errors)
        temp = hidden_errors * hidden_values * (1.0 - hidden_values)
        temp = self.learning_rate * temp

        delta_weights_ih = np.dot(temp, inputs.T)
        self.weights_to_hidden = np.add(self.weights_to_hidden, delta_weights_ih)
```

I also have code written for my take on the word2vec technique. In this occasion I made use of the word2vec implementation of the genism library. However I wrote my own code for reading the text from a file and made use of that class to create the input to that word2vec model from genism. I intend for this mini implementation of the word2vec to be incorporated into the main project as it is the building block of an actual implementation.

My file handler class:

```
class FileHandler:

    def __init__(self, text_file="TextFiles", saved_models="SavedModels"):
        """
        A class to handle file and text manipulation for the Word2Vec project. If passed in
folder names
        do not exist, the folders are automatically made

        :param text_file: name of folder to store dataset
        :param saved_models: name of folder to store saved word2vec models
        """

        self.__path = os.path.dirname(os.path.abspath("."))
        self.text_file_path = os.path.join(self.__path, text_file)
        print(self.text_file_path)
        self.model_file_path = os.path.join(self.__path, saved_models)
        self.sentence_processor = SentenceProcessor(self.text_file_path)

        if not os.path.exists(self.text_file_path):
            os.mkdir(self.text_file_path)

        if not os.path.exists(self.model_file_path):
            os.mkdir(self.model_file_path)

    def print_to_file(self, file_name: str, content: str):
        """
        Prints to file
        :param file_name: name of file to be created
        :param content: string to be printed to the file
        """

        f = open(self.text_file_path + "\\" + file_name + ".txt", mode="w", encoding="utf-8")
        f.write(content)
        f.close()

    def get_file_contents(self, filename: str):

        with open(os.path.join(self.text_file_path, filename), "r", encoding="utf-8") as f:
            read_data = f.read()

        return read_data

    def get_sentence_processor_iterator(self):

        return self.sentence_processor
```

In the main function:

```
if __name__ == "__main__":

    file_handler = FileHandler("TextFiles")

    corpus = file_handler.get_sentence_processor_iterator()

    model = Word2Vec(corpus, iter=50, min_count=15, size=300, workers=4, window=10)

    # Print the words that are the most similar to the word "software"
    print(model.wv.most_similar(positive="software"))
```

4.2. Plan of Future Work

Below is the general plan I have for the remaining time I have for the FYP. I cannot fully create a concrete and accurate plan as I cannot predict the unexpected obstacles that may come my way. I will however outline what is left to do and the optimal time I can accomplish them. The report will be worked on throughout the lifetime of the FYP.

2nd – 21st Jan. 2019 (Remaining holidays)	• **Find and collect dataset for training and testing classification model** • **Start development of main product** ○ Experiment with GUI libraries for Python ○ Make a start on the "business logic" of the application including the word2vec/doc2vec and the classification model ○ If time still available, plan the design of the overall system and contemplate possible design patterns to include
21st Jan. – 28th Mar. 2019 (Semester 2)	• **Complete development** ○ Apply plans and designs created during the holiday ○ Integrate each component of the software system together • **Test & Evaluate Classification Model** ○ Create and use test cases ○ Evaluate accuracy of the classification model ○ Adjust model parameters to improve accuracy

References

Aggarwal, C. C. & Zhai, C., 2012. A Survey of Text Classification Algorithms. *Mining Text Data,* January.pp. 163-222.

Brownlee, J., 2016. *Supervised and Unsupervised Machine Learning Algorithms.* [Online]
Available at: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/
[Accessed 20 November 2018].

Klein, B., 2011. *Backpropagation in Neural Networks.* [Online]
Available at: https://www.python-course.eu/neural_networks_backpropagation.php
[Accessed 16 Oct 2018].

Korde, V. & Mahender, N., 2012. Text Classification and Classifiers: A Survey. *International Journal of Artificial Intelligence & Applications (IJAIA),* 3(2), p. 85.

McCormick, C., 2016. *Word2Vec Tutorial - The Skip-Gram Model.* [Online]
Available at: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/
[Accessed 20 October 2018].

Mikolov, T., Chen, K., Corrado, G. & Dean, J., 2013. *Efficient Estimation of Word Representations in Vector Space.* Arizona, Proceedings of Workshop at ICLR.

Mikolov, T. & Le, Q., 2014. *Distributed representations of sentences and documents.* Beijing, JMLR.org, pp. 1188-1196.

Nielsen, M., 2018. *Using neural nets to recognize handwritten digits.* [Online]
Available at: http://neuralnetworksanddeeplearning.com/chap1.html
[Accessed 15 Oct 2018].

Rashid, T., 2016. *Make Your Own Neural Network.* Germany: CreateSpace Independent Publishing Platform.

Rosenblatt, F., 1958. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review ,* 65(6), pp. 386-388.

Stergiou, C. & Siganos, D., n.d. *Neural Networks.* [Online]
Available at: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#References
[Accessed 24 December 2018].

TensorFlow, 2018. *Vector Representations of Words.* [Online]
Available at: https://www.tensorflow.org/tutorials/representation/word2vec
[Accessed 17 October 2018].