Using a Recurrent Neural Network model to generate Hip Hop lyrics

UNIVERSITY *of* LIMERICK

OLLSCOIL LUIMNIGH

Department of CSIS

FINAL YEAR PROJECT - 2018

By: **James Daly**

ID: 14152789

Bachelor of Computer Games Development (LM110)

Supervised by: **J. J. COLLINS**

Abstract

The aim of this project was to mirror the quality and complexity of Hip Hop artists lyrics through the use of Recurrent Neural Networks(RNN) and the Long Short-Term Memory(LSTM) model. The RNN and LSTM model is a form of Artificial Neural Network used in modern day machine learning for creating predictive models. In this study the model will be tuned on a large body of Hip Hop lyrics. Once adequately trained the model will respond to user input with an output of a predefined length. Hip Hop lyrics were chosen for their complex rhyming schemes and adherence to structure. The findings in this paper demonstrate the complex nature of producing artistic work through the machine learning paradigm. It also shows the interesting nature of the framing problem and it's repercussions on the field of AI produced art.

Declaration

I declare that this is my work and all contributions from other persons have been appropriately identified and acknowledged.

Contents

Chapter 1

Introduction

1.1 Summary

This paper is about the applicability of current machine learning approaches to the problem of writing Hip Hop lyrics. I examine current approaches being used for the purpose of replicating human creative writing and propose my own solution to the task. I also provide a novel approach to the objective evaluation of lyrics.

Hip Hop or rap as it is often referred to is a large genre of music which originates from the densely populated urban areas of the USA. The reason I selected this genre of music for replication was twofold. The fist reason is that I simply have a passion the genre. Secondly, I believe it's strong adherence to structure and rhyme makes good rap lyricism easier to objectively quantify than many other genres who's beauty is often derived acoustically or from the individual artist's vocal range. In hip hop lyrics there is a concept called bars. Bars are small groupings of lines or stanzas usually containing 4 lines. These lines together form a rhyming scheme which may vary from bar to bar. An example of a bar is:

> Please reply to me Weenz...

The problem of synthesising hip hop lyrics is a complex one. At its core there is the field of Natural Language Processing(NLP). NLP is " an area of research and application that explores how computers can be used to understand and manipulate natural language text or speech to do useful things"Chowdhury (2003). Natural language is described as "one developed and evolved by humans through natural use and communication , rather than constructed and created artificially, like a computer programming language"Sarkar (2016).This is often viewed as problem of decomposition, breaking blocks of text into sentences, then into phrases and finally into individual words so as to understand the structure and ultimately, the meaning of said text.

In recent years there have been many attempts at replicating human rhyming schemes. Some promising work has been done in the field using Hidden Markov Models Barbieri et al. (2012). These approaches use Markov Models to build a probabilistic estimate of the next letter to be used in a sentence. These methods required large amounts of human interference in terms of constraint modelling. Other research has been done in the area using methods like Rank Support Vector Machines Malmi et al. (2015). These techniques choose appropriate lyrics from large body of previous works. While these methods have produced believable lyrics they do so by copying lines from other artists wholesale.

In this paper I look at the use of Neural Networks for solving the problem of imitating hip hop lyrics. A neural network is a collection of neurons which may be organised and trained to perform tasks. A neuron is a cell in the human brain which may receive multiple inputs and produce a single output. A neuron may be replicated through data structures and algorithms in software to create an Artificial Neural Network. Artificial Neural Networks can be trained to store large amounts data and apply that data to a problem, in this case synthesising hip hop lyrics.

The type of neural network I use in this paper is called a Recurrent Neural Network. This is a derivation of the Artificial Neural Network which contains feedback loops. This allows for a form of history and a concept of time which does not occur in more traditional Convolutional Neural Networks. This allows

for the creation of continuous output meaning that the network may produce an infinitely long output if required. In this case that means that the network, once given an input, may produce lyrics indefinitely.

I use a derivation of the Recurrent Neural Network called the Long Short-Term Memory model. Each neuron is given its own piece of memory which allows the network to form longer temporal relationship. This allows the network to produce lyrics with rhymes and structures which play out over several lines, stanzas and verses.

This network is then applied to an adversarial system. In hip hop or rap there is a concept called the rap battle. It is an adversarial game in which two participants attempt to outperform each other by rapping. The competition is usually judged on metrics such quality of wordplay, rhyme scheme and offensiveness of insults. This network once trained can be loaded in a framework allowing a user to engage in a rap battle with the network itself.

I also propose a new approach to automated evaluation of the lyrics produced. I created an evaluation suite to inspect for spelling mistakes in the text, rhyming density and for semantic consistency. The presence of automated evaluating allowed for faster prototyping and an objective measure for testing output. This provides a clear and concise measure for comparing the output of different systems.

I show in this paper the power of the Recurrent Neural Network and the Long Short-Term Memory model in producing Hip Hop lyrics. I also outline and develop a model for evaluating these lyrics and show the benefits of having an objective, automated and streamlined process for evaluation.

1.2 Objectives

The primary objectives are threefold:

1. The main objective of this paper is to show that through the use of a Recurrent Neural Network combined with the Long Short-Term Memory(LSTM) model Hochreiter and Schmidhuber (1997) a computer can produce lyrics with the same level of musical complexity and coherence as a human hip hop artist. To show the benefits of LSTM and how it helps to tackle the exploding/diminishing gradients problem. Pascanu et al. (2012)
2. To provide a framework for automated testing of lyrics.
3. Show the impact of different parameters on the network and the effect they have on the quality of lyrics produced.

As this project covers such a large pair of topics, machine learning and musicology , there are many secondary objectives that are tackled in this paper:

- Implementing and training a Recurrent Neural Network
- Learning about the deep learning paradigm
- Learning about second order optimisations for neural networks
- Study of Long Short-Term Memory models
- Mathematically quantifying rhyme
- Mathematically quantifying coherency
- Exploring the limits of AI produced art

1.3 Contributions

I show how this form of Network may be used to create an adversarial system with which a human may interact. I also create a suite of tools and training sets which may be used to further fuel research in this field. In this paper I highlight the effects of different training sets on the training of the network. I show how training sets derived from a list artists has a significant impact on the quality of works produced.

1.4 Overview of the Report

This report is broken into a few major sections.

Introduction This section contains a basic overview of the project, objectives, motivations and a brief discussions on other contributions related to this project.

Background Research This section discusses the findings of existing research in the area of text synthesising using machine learning techniques.

Design and Implementation This section demonstrates in detail the implementation of the Recurrent Neural Network, the Long Short-Term Memory Model and the various tools developed to quantify and test lyrics produced.

Empirical Studies This section discusses the various tests and pieces of analysis done on the outputted as well as the variable performance of differently parametrized networks.

Discussion and Conclusions This section contains the discussions on findings in empirical studies and conclusions.

1.5 Motivation

Whilst studying computer games development I started to create very basic games. At first these games were simple but over time as their complexity grew I began to add more and more systems. I began to experiment with very rudimentary AI systems. As I dove deeper into the topic of AI I began to see some of the ground breaking work being done in this field as of late.

I soon came across the interesting field of neural networks. I had seen how these network's had been used to create AI's which could drive cars, play games and even create rudimentary art. The art aspect really grabbed my attention as it seemed like a medium in which computer science should have no stake.

From what I had read online the easiest art form to imitate through AI was that of generating creative writing. The most common system used in this niche field was that of recurrent neural networks. I began to take a small course in the area but soon became bored. The tutorials I was following at the time were using the works of Shakespeare. While I am not entirely against the great bard as a person I found recreating works in his style immensely boring.

I looked for ways of making this work more interesting. I decided that the best remedy for the situation would be to find an author I was more interested in. I had seen talk of a similar project using Bob Dylan and decided I would attempt something similar using artists that I am interested in. It was at this point that I decided to attempt a recreation of the works of my favourite hip hop artists.

Hip hop as a genre and a culture is something I am deeply passionate about. When I was younger I empathized strongly with the works of many prominent rap artists as they spoke about a generation of their peers lost to gang violence and prison sentences as I lost a whole generation of men in my local area to suicide and emigration. This empathy paired with a keen interest in the complex writing mechanics used in rap makes hip hop a genre I'm passionate about.

After researching online I found a limited body of work which had been undertaken in the area. I decided that I would very much like to expand upon this work. I felt it would be a great opportunity to learn more about the field of AI and with some luck I could discover more about the science of rap along the way.

In conclusion I have undertaken this project as a chance to build upon the work which has been done in the area of computer generated lyrics by building upon the system used in *Ghostwriter* Potash et al. (2015) . I also provide a template for anyone looking to push this work forward in the future.

Chapter 2

Background

2.1 Introduction to Machine Learning

Machine learning is a sub field of computer science that is concerned with creating programs and systems that have the ability to learn and improve over time. This learning may be performed by iterating over a large body of data. Machine learning is a very large field in and of itself and contains many subsections. Machine learning as a term was first coined by Arthur Samuel in 1959 whilst working at IBM. Arthur Samuel described machine learning as a,

"Field of study that gives computers the ability to learn without being explicitly programmed" Munoz (2014).

Machine learning relies upon large bodies of data over which it may iterate to create groupings, find patterns and extrapolate concepts.

Machine learning as a field has existed since the early days of computer science. Machine learning first developed as an offspring of the quest for artificial intelligence. Early AI systems were often designed as finite state machines. This is evident in the classical expert systems built throughout the early days of computing. Expert systems were constructed as a series of logic branches in which the engineer would encode a set of predefined rules. All the information and logic these early AI systems would use was encoded during the design phase by a programmer. This meant that these early AI systems had no ability to

improve themselves and could only be improved by a programmer rebuilding them with extra layers of logic. This approach to AI has two major flaws. As programmers add more and more logic, the complexity of the underlying code can explode and become untenable. This will make it harder and harder to improve the AI over time as more and more edge cases are added. There is also the problem of cyclomatic complexity. This is the problem of code becoming ever more branching over time which leads to a maintainability and an extensibility nightmare. Wilkins (2014)

On top of this issue of software complexity there is also the huge problem of tasks humans cannot solve. With classical AI systems the programmer needs to encode all the rules and information within the system. A problem arises then when you want to craft an AI to solve problems that humans don't have a solution to Singh (2010). If the designer does not know a rule she cannot encode it. The remedy to this solution is to create systems that can learn and then discover solutions to problems humans could not solve. This gets around the issue of designers not knowing the rules to encode as a machine learning may extrapolate and encode its own rules.

Machine learning contains many branches. Each with their own strengths and weaknesses. These branches can be divided into two main categories, unsupervised and supervised learning. Unsupervised learning is the task of inferring a function or solution from unlabeled data. This means that a system must extrapolate its own concepts and groups from the data set fed into it. An example of unsupervised learning is that of clustering. This is a system where by unlabeled raw data is plotted across high dimensional space to form clusters and groupings. The idea behind cluster analysis is to create and outline distinct groupings within a dataset allowing for future data to be classified. With unsupervised learning there is no need for the person training the system to know the correct answer to the question being posed to the network.

The category this project falls into is that of supervised learning where all data will be tagged. This will allow the system to see the specific patterns found inside of the data and create a predictive model. An example of supervised learning is that of an Artificial Neural Network using back propagation.

With back propagation a network is fed in some data and then gives an output.Munoz (2014) The output is then compared with the expected output for that input and an adjustment is made that will propagate backwards through the network. Supervised learning requires whoever is training the system to provide a correct output for every input.

The subsection I take on in this project is the area of neural networks and their applicability to language modelling.

2.2 Machine Learning Paradigms

2.2.1 Recurrent Neural Networks

Recurrent Neural Networks are currently used heavily in the field of Natural Language processing Sutskever et al. (2011). More specifically it is currently being used in the field of creative writing Ghazvininejad et al. (2016).

Current Recurrent Neural Network approaches are mostly based in the character level. This means that these systems produce their bodies of text one letter at a time Sutskever et al. (2011). These systems provide a high level of flexibility in their ability to produce lyrics but often make spelling and grammer mistakes Potash et al. (2015). They also struggle with long term dependencies with lines often having no real relationship with each other Li et al..

An RNN is a form of artificial neural network. This means that the system is a large network of neurons containing an input and output layer. A neuron is a structure found in nature that makes up the brain. It uses electrical charges to either block or fire other neurons Rosenblatt (1958). These neurons in turn may be fired by inputs, such as the human eye, or may fire a nerve for output, such as a muscle tendon nerve. A neuron is made up of three main parts. The dentrites are an area which accepts inputs, the cell body which takes the inputs and modifies them, and the terminals or synaptic region which outputs the result. See figure 2.1 of a real neuron.

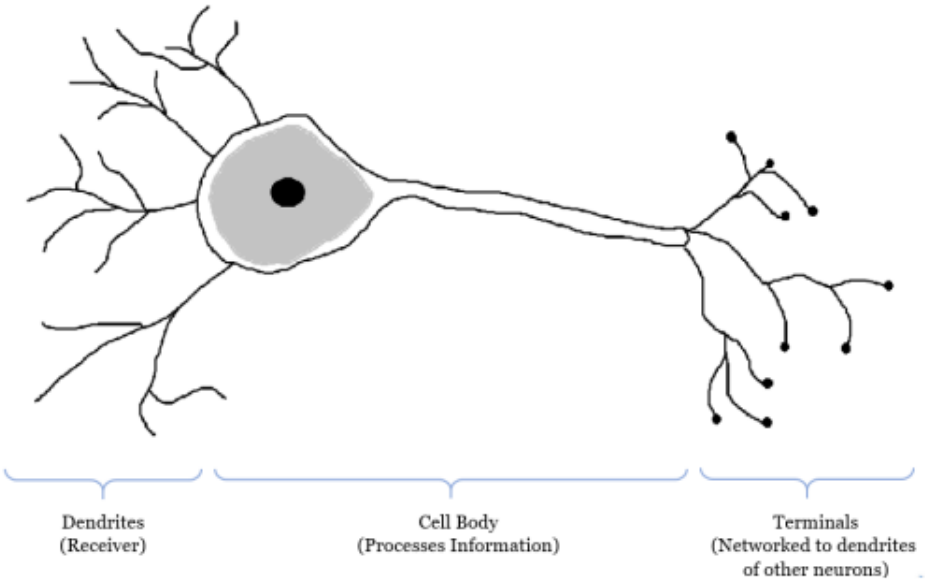


Figure 2.1: A modified diagram of a neuron taken from Granville (2016)

History of the Artificial Neural Network

When computer scientists first studied the field of artificial intelligence they believed that the most difficult problems in computing would be the tasks that humans find difficult ie. playing checkers at a high level or planning a route on a large and complex map. As it turned out these problems were almost trivial in nature. The extremely difficult problems in the field of AI were the things that humans often took for granted such as identifying a human in a picture or speaking a language.

Some researchers soon realized that for certain problems the best way to imitate human intelligence was to imitate the human brain. This is the core principal behind the artificial neural network Rosenblatt (1958). The basic idea was simple, a brain is a large cluster of rapidly firing neurons. On their own merit, neurons are very simple and straight forward units which accept a collection of inputs and produce a single output. As time went on research in AI began to build on the core idea of the perceptron creating many derivatives which helped to solve differing and interesting problems. First there was The Adeline Widrow and Hoff (1960), this brought about the concept of a trainable system. Later researchers began to realize the weaknesses of the single layer approach Minsky and Papert (1969) and soon multilayer neural networks became the norm.

For a long time this research would remain firmly planted in the world of academia. Computing power was not in a place where deep and complex neural networks could be run on ordinary hardware. The large structures of these neural networks and their inherently parallel nature meant that they were not well suited for the single core, sequential operation machines of the 20th century. It wasn't until the mid 1990's that neural networks found their first practical usage. Neural networks were first used to recognize cursive writing on cheques Dodel and Shinghal (1995). This was the first time ANNs were used in a commercial product as this software was rolled out in banks right across the world.

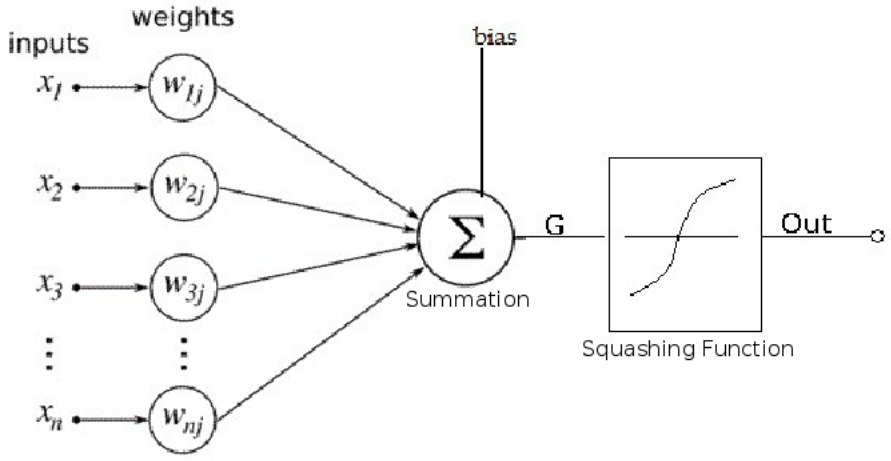


Figure 2.2: A modified diagram of a neural network taken from Sau (2015)

How the Neurons Work

An artificial neural network mimics this. In a natural neural network information is passed via the exchange of electric charge and travels at a speed of roughly 3m/s Rosenblatt (1958). In an ANN this is replicated by the use of scalar values and theoretically has no such speed limitations. In an artificial neuron the dentrites are swapped out for inputs. These inputs are then given weights. This is to imitate the fact that in a natural brain the different neurons impact on each other is mutated by the level of conductivity between neurons. Artificial neurons then take these weighted inputs and sum them. The summed figure is then passed through a function where the value is mutated. This mutated value is then passed through a squash function that usually squashes the number to bound the output between 0 and 1 or 1 and -1. This newly squashed number is then passed on to the next neurons. A mathematical representation of how a neuron works can be seen in 2.2.

From here on in when a neuron is mentioned it may be assumed that I am talking about an artificial neuron. Every neural network is built of a collection of these neurons that have been interconnected in a specific formation depending on the type of neural network. I will only be focused on multilayer neural architectures. There are three main parts to a multi-layered neural network.

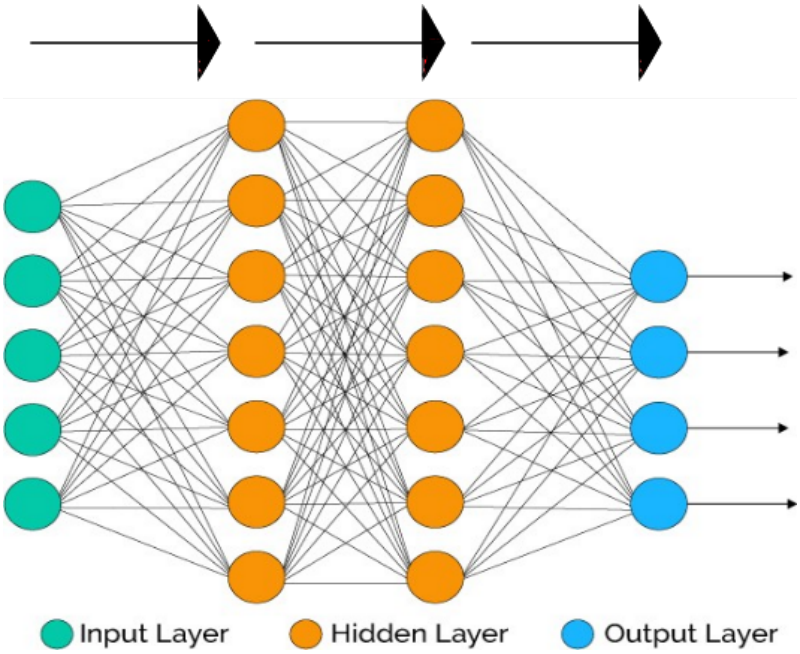


Figure 2.3: A modified neural network image taken from this article http://technicalsurfing.blogspot.ie/2017/12/artificial-neural-networks.html

The first is the input layer. This layer is only one neuron deep but may be as many neurons across as required. This layer, as the title explains will accept all the input. The input of these neurons will be external data. The next layer in the network is the hidden layer. The hidden layer may vary in size between systems. The hidden layer may be as deep and as wide as is required for a particular task. The hidden layer is where most of the work is done. The size of the hidden layer will determine just how hard a system is to train and how much hardware will be required but it is also the hidden layer that dictates the scope of the systems abilities. The final layer is that of the output layer. The output layer is similar to the input layer but instead of taking in input from the external world it sends its output to the external world and its neurons are fired by the hidden layer. A multilayer network can be seen in **??**.

The Recurrent Neural Network specifically

The specific type of neural network I am using for language modelling is a recurrent neural network. In the figure **??** you will notice that there are no vertical lines. This means that all the information in the network only ever flows forward. This means that information is fed in once at a specific size and then an output, the size of the output layer is produced. In a recurrent neural network there are loop backs and interconnections in layers see figure 2.4. This has many great benefits. This creates a concept of history within the network where work done further down the network is effecting the work being done in a layer now. The network can now accept continuous input and give continuous output. This introduces timesteps, meaning that every time the neurons are fired new input can be given and more output is produced.Hochreiter and Schmidhuber (1997)

This combined with history allows the network to have its own idea of forward planning. It is this forward planning and awareness of the the past that makes this the perfect architecture in my eyes for natural language processing.

2.2.2 How Recurrent Neural Networks learn

Recurrent Neural Networks learn through a process called back propagation. This process is done through passing an input into a network, getting an output and then adjusting the weights of the networks connections relative to the difference between the output produced and the output expected Hinton (1992). First the error of the network is computed by the formulae;

1. j is a unit in the output layer
2. y is the actual output
3. d is the desired output
4. E is the error
5. k number of neurons in layer

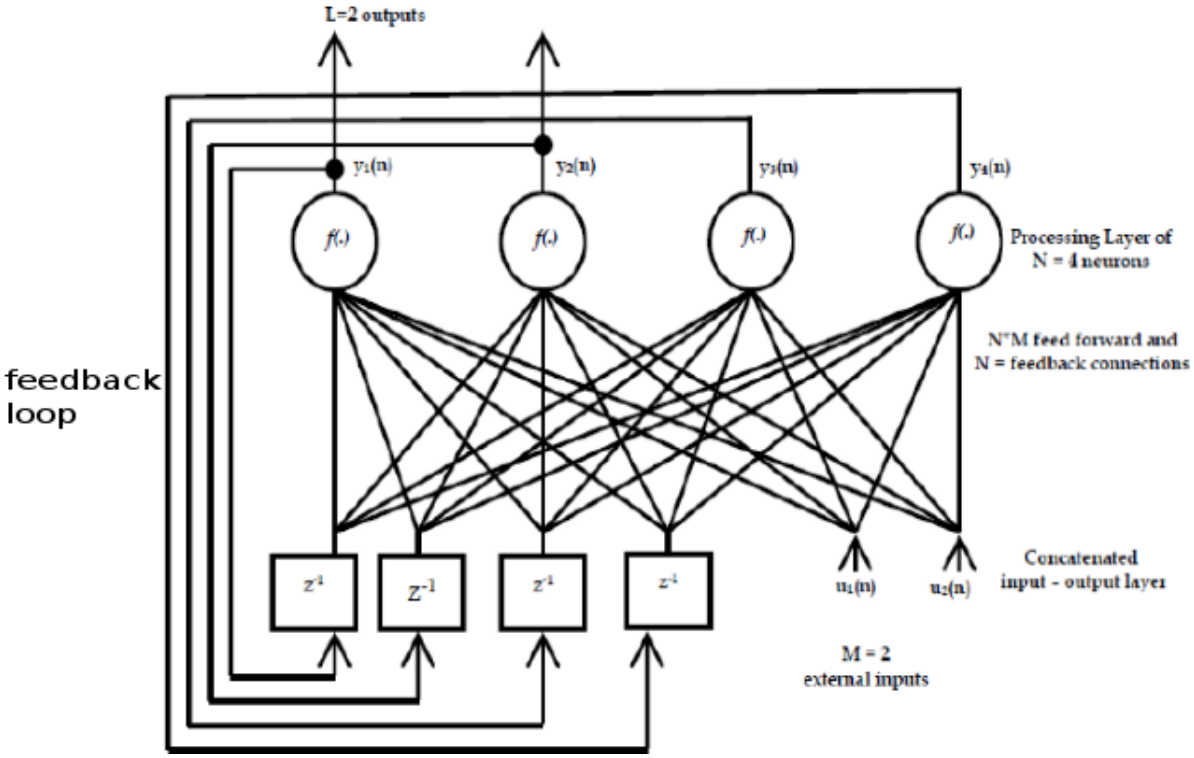


Figure 2.4: A modified diagram of an RNN taken from http://the8es.co/neural-network-example/example-of-a-fully-recurrent-neural-network-of-type-1-dijk-1999-4f8a3846.html

$$E = 1/2 \sum_{j}^{k} (y_j - d_j)^2$$

From here a desired amount of change is calculated for each individual weight using the selected formula for the network. This change is pushed back down the network the same way input is pushed forward being manipulated by weights along the way. Hinton (1992)

The Hopfield Model approach involves this value as an energy function. The energy function of the network is calculated and then weights of connections between neurons are adjusted using the following formula;

1. $u_i(t)$ is the output of the neuron i at timestep t
2. f_n is a hard limiting non linearity function which changes the output to a binary one
3. t_{ij} is the connection weight between neuron i and j

$$u_i(t+1) = f_n[\sum_{i=0}^{N-1} t_{ij} u(t)]$$

The weights of each connection are adjusted until $u_i(t+1)$ converges on the expected output Lippmann (1987). Convergence is defined as when changing of any of the weights no longer decreases the energy function of the network Lippmann (1987).

This process of changing weights is known as gradient descent. Every neural network may be viewed as a function mapped across a multidimensional space. The weights of the network are being tuned to find a local minima of this function. The local minima refers to the lowest possible error rate averaged across all possible inputs and outputs.

2.2.3 SVMs

Support Vector Machine(SVM) was another approach used for synthesising lyrics. The specific form of SVM used for creating hip hop lyrics is that of the

RankSVM Malmi et al. (2015). This form of Support Vector Machine doesn't just categorise data points but actually ranks them. In the *Dopelearning* paper a huge body of text, some 500,000 lines of lyrics Malmi et al. (2015), is provided to the system which may be ranked according to rhyme and provided as a response to each preceding line of text. Whilst this process did provide strong rhyming and good grammar it suffered from semantic inconsistencies merely parroted other's work in a strange patch work of writing styles.

The Support Vector Machine Algorithm is a mathematical function designed to find the best possible hyperplane to bisect two separate groupings in a population. An example of one these neurons can be found in figure 2.5. The core concept is to find the plane with the largest possible margin between itself and the two groupings.**?**. This is a form of supervised learning where the algorithm will be told which grouping the data points relate to.

Support Vector Machines are a different sub section of machine learning. They were developed in the reverse order of the Artificial Neural Network which the RNN is built upon. Neural Networks followed a heuristic path from extensive experimentation first followed by theory later, whereas Support Vector Machines were first theorised as mathematical models and then later experimented with in the computer science community Wang (2005). As a result of this the Support Vector Machine went unnoticed with the computer science community for a long time.Abe (2006)

Support Vector Machines have many advantages and disadvantages over Neural Networks:

Advantages

1. With a Neural Network the generalization ability deteriorates greatly as the size of the training set gets smaller. This deterioration effect is greatly reduced in Support Vector Machines.Abe (2006)

2. Because Support Vector Machines use a quadratic equation and not a high level polynomial they have only one global minima unlike Neural Networks which have several local minima.Winston (2014)

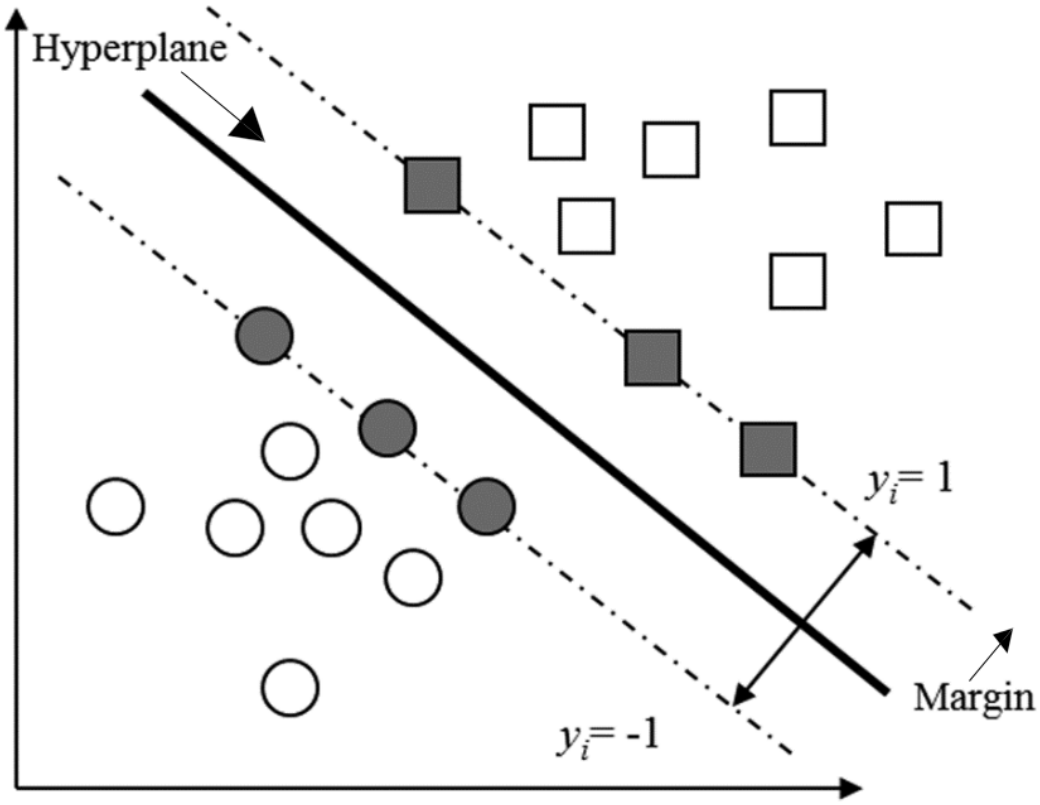


Figure 2.5: A modified diagram of an svm drawn hyperplane taken from http://www.flughafen.asia/learning-classify-machines-international-engineering.html

3. Support Vector Machines are much less vulnerable to outliers in the training set.Abe (2006)

Disadvantages

1. Support Vector Machines use direct decision functions making extension to multi-class problems more difficult.Abe (2006)

2. They also have a much higher training time and larger memory footprint than many common Neural Network Models.Abe (2006)

3. Support Vector Machines do not free you of some of Neural Networks main problems. Chief among them is the issue of parameter selection which may be a formidable task for even experienced researchers.Abe (2006)

Support Vector Machines may be used to group words or sentences together by rhyme. Whilst this approach does produce coherent writing and conforms to rhyming schemes it also means that it will never produce an original line of text. This would be extremely frowned upon within the artistic community. It would also greatly diminish the artistic value of the work. It was for this reason I decided not to use a Support Vector Machine approach.

2.2.4 Hidden Markov Models

Hidden Markov Models were yet another approach I considered. Hidden Markov Models have been used in the field of generating lyrics using Constrained Markov Processes Barbieri et al. (2012). In the paper *Markov Constraints for Generating Lyrics with style* the authors used Markov Models to generate lyrics in the style of a specific author using their corpus of work. The constraints in this process were extremely tight and created syntactically correct writing but it was concluded that the structural properties of texts demand long distance modelling that Markov Processes can not provide.

Hidden Markov Models are a statistical Markov Model where it is assumed that the system being modelled is a Markov process with hidden states. The algorithm looks at the previous entries in a sequence to determine the probability of the next entry. It then picks the next entry at random, weighted for their relative probability. Jurafsky and Martin (2014)

Whilst this technique could manage to produce coherent lines of text it would struggle with long temporal relationships. This is because Hidden Markov Models struggle with long term dependencies resulting in lines having very little to do with each other.Barbieri et al. (2012) It is this lack of long term dependencies which made me shy away from this method of generating text. This is because the rhyming structure of Hip Hop lyrics is often built over the course of several lines and even verses. This means that a form of long term memory is needed.

Chapter 3

Design and Implementation

3.1 Introduction to Software

For this project a suite of software was produced to tackle a wide range of issues and tasks. These tasks ranged from the automated testing of outputs to the collection of training data. All code was written in Python. Some code was produced in Python 2.7 whilst other parts were written in the newer more up to date Python 3.5. The decision to use two different versions of the same language was a tough one but it was ultimately made because of a lack of library support for both languages. If this study were to be expanded upon and improved it would be highly recommended to take the time to port all code over to Python 3.x as many of the aforementioned libraries are open sourced.

The software tools I created for this project are as follows:

1. Spellyzer
2. Lyric Scrapers
3. Lyric Changer
4. Rap Battler

3.1.1 Spellyzer

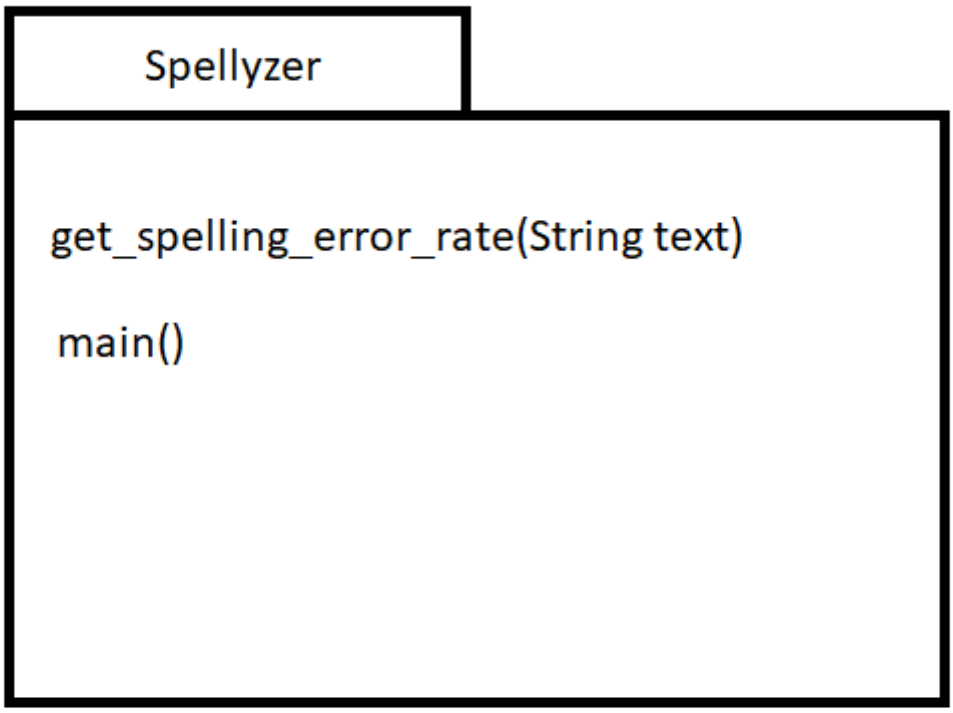
The Spellyzer is a piece of software created to allow a quick and automated way to test the quality of output from the neural network. The job of the Spellyzer is to check the spelling ability of the network. It also checks how many words were produced. This statistic is useful when testing the network as number characters per word may be used as a proxy for the networks ability to form longer words.

The spellyzer is a short script written in Python 3.x using the enchant and regular expressions libraries for Python. The code first splits all of the text into individual words which may be seen on line 2. Regular expressions are then used to remove all non alphanumeric symbols that may have been mistaken for words such as an exclamation or question marks. This regular expression may be seen on line 1. The enchant library is then used to see if the words remaining are valid words found in the US-English dictionary. This check is preformed on line 10. This dictionary was selected as all of the training data used was extracted from American artists.

```
1  regex = re.compile('[^a-zA-Z0-9]')
2  words = text.split()
3  dictionary = enchant.Dict('en_US')
4  total_words = len(words)
5  print('Total words       : ' + str(total_words))
6  incorrect_words = 0
7  for word in words:
8    word = regex.sub('', word)
9    if(word):
10     real_word = dictionary.check(word)
11     if(real_word == False):
12       incorrect_words += 1
13 print('Incorrect words   : ' + str(incorrect_words))
14 error_percentage = (float(incorrect_words)/float(total_words))
    *100
15 print('Error percentage  : ' + str(error_percentage) + '%')
```

This script was developed so that it could be run on it's own as a standalone script or could be accessed and used by other software. In this project the

Spellyzer was accessed externally by the program which produced lyrics. This meant that every time the user ran the Neural Network the output was accompanied by a small print statement indicating the spelling error of the output. This sped up the testing cycle for the study greatly.



If given more time this system could be expanded to include a more attractive and user friendly interface, possibly containing visual representation of the spelling mistakes. This software could also be expanded to account for near misses where a word was close to be spelled correctly but just off ie. Hello -> Hallo

3.1.2 Lyric Scrapper

The Lyric Scrapper is what made this project possible as it allowed me to create a large training set of lyrics quickly and easily. This software was written to scrap a website named lyricsfreak.com lyr. The software went through a list of artists, navigated their page on the site and opened each individual song. Each song was scraped from its respective page and written out to a file. A file containing the artists entire corpus was also created. The Lyric Scrapper could be configured to produce two different file types, one containing the lyrics in plain text and a second file containing all the lyrics with special tags to indicate new lines and different paragraphs.

Lyric Scraper was written in Python 3.x and leaned heavily on the Beautiful Soup library. This library is used for parsing HTML and XML. It creates a parse tree making it easier to navigate through the page programmatically. A parse tree is first used to navigate the artist's page to find links to their individual songs. Once the individual webpages of these songs are found they are downloaded and converted into parse trees also. These new parse trees are then navigated and scrapped for lyrics. This was a large difficult in the project as the HTML pages for these songs were not written with scrapping in mind and as such contained many strange edge cases and were less than intuitive to scrap lyrics from. These lyrics, once retrieved, were then written to text files. These files are organised by artist.

Below is a code snippet showing all of the artists used and their corresponding URLs.

```
1  artist_names = ['kendrick_lamar/',
2    'aesop_rock/',
3    'j_cole/',
4    'scroobius_pip/',
5    'lil_wayne/',
6    'tupac/',
7    'nas/',
8    'dre/',
9    'eminem/',
10   'kanye/',
```

```
    'rakim/',
    'outkast/',
    'fabolous/',
    'busta_ryhmes/',
    'jay-z']

artist_URLs = [ 'http://www.lyricsfreak.com/k/kendrick+lamar/',
    'http://www.lyricsfreak.com/a/aesop+rock/',
    'http://www.lyricsfreak.com/j/j+cole/',
    'http://www.lyricsfreak.com/d/dan+le+sac+vs+scroobius+pip/',
    'http://www.lyricsfreak.com/l/lil+wayne/',
    'http://www.lyricsfreak.com/t/tupac+shakur/',
    'http://www.lyricsfreak.com/n/nas/',
    'http://www.lyricsfreak.com/d/dr+dre/',
    'http://www.lyricsfreak.com/e/eminem/',
    'http://www.lyricsfreak.com/k/kanye+west/',
    'http://www.lyricsfreak.com/r/rakim/',
    'http://www.lyricsfreak.com/o/outkast/',
    'http://www.lyricsfreak.com/f/fabolous/',
    'http://www.lyricsfreak.com/b/busta+rhymes/',
    'http://www.lyricsfreak.com/j/jay+z/']
```

Each artist in the list was looped through and their lyrics scraped.

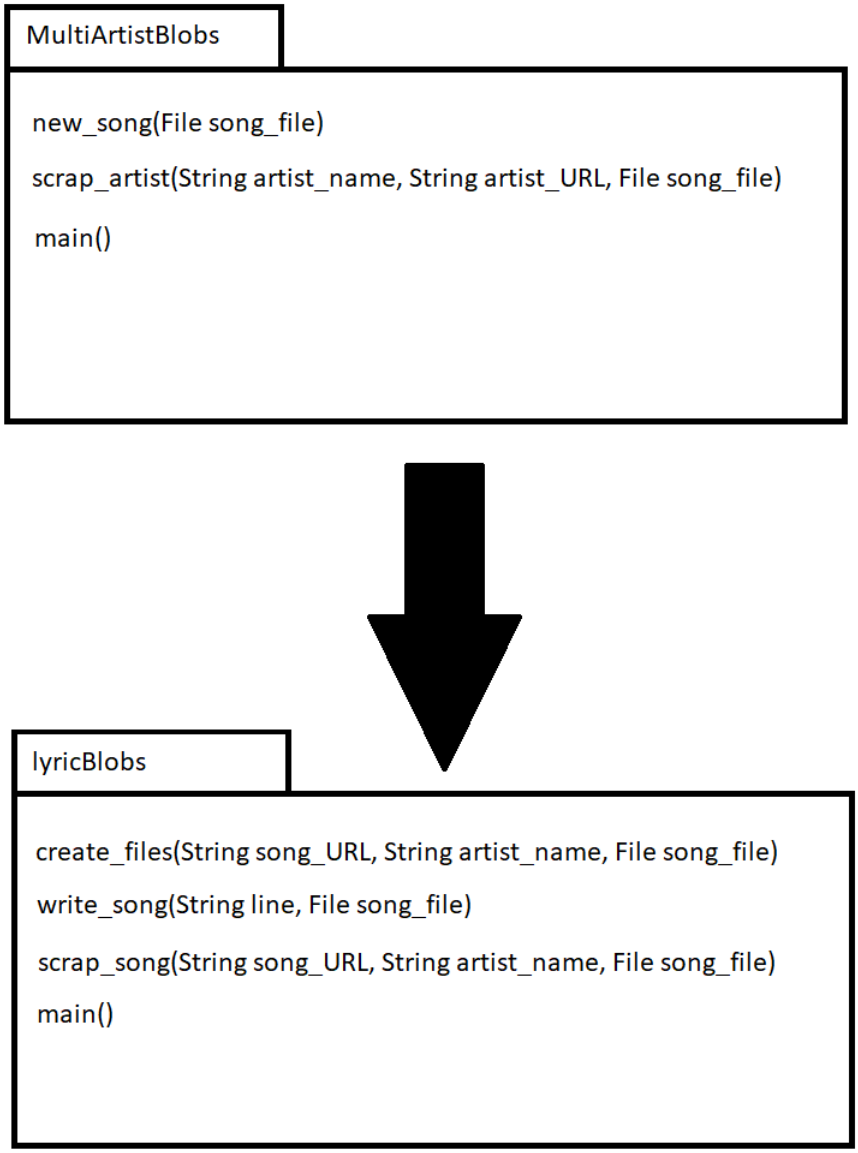
```
for i, artist in enumerate(artist_names):
  try:
    scrap_artist(artist, artist_URLs[i], song_file)
```

Each page was individually scrapped of all its lyrics. On line 4 the page was scrapped and on line 5 the html of that page was turned into a parse tree with beautiful soup. The parse tree is then carefully navigated using specific steps for processing the website chosen. This means that this lyric scrapper only works for this site.

```
def scrap_song(self, song_URL, song_id, artist_name):
  opener = urllib.request.build_opener()
  opener.addheaders = [('User-Agent', 'Mozilla/5.0')]
  song_page = opener.open(song_URL)
  song_soup = BeautifulSoup(song_page.read(), 'html.parser')
  song_page.close()
```

```
7       lyrics = song_soup.find('div', {'id':'content_h'})
8       song_file = self.create_files(song_URL, song_id, artist_name)
9       first_line = lyrics.next
10      log = []
11      if(first_line.string[:1] != '['):
12        log.append(first_line)
13        lines = lyrics.find_all('br')
14        for line in lines:
15          current_line = (line.next).string
16        if(current_line == None):
17          log.append('\r\n')
18        elif(current_line[:1] == '['):
19          pass
20        else:
21          log.append(current_line)
22          for entry in log:
23          self.write_song(entry, song_file)
```

This software was developed in modules. These modules cover a range of tasks. There are two modules for scrapping individual artists both tagged and untagged. Then there is another module named Multi Artist Blob which coordinates these modules and can be configured to scrap several artists at a time. This approach allows for extra flexibility for future developers who may only want to scrap specific songs and not whole artists.



In hindsight this software also could have done with a graphical user interface to make it easier to interact with. Also an extension or ability to scrap songs from other websites or sources would have been a welcome addition to the software.

3.1.3 Lyric Changer

This piece of software is intuitively named Lyric Changer. It's sole purpose is to take the lyrics or transcripts of songs by artists and to remove everything that is not a lyric ie. (Chorus x3) and identify bars in the song. Bars are a Hip Hop term used to describe tightly linked lines in a song. They are delivered in groups of 4. The job of the Lyric changer is to find verses which contain a multiple of 4 lines and break them into 4 line bars. These bars are then shuffled and written to a file. This process removes noise from the dataset and leaves a clean file containing only rappers bars which makes for a much stronger training set.

This software uses only modules found in the Python standard library. It is configured to go through a list of artists directories that have already been downloaded and saved by the Lyric Scrapper. It accesses each artist one by one and extracts their lyrics. All non lyrically valid lines are discarded and then all the verses are scoured for bars. These bars are then all shuffled and written to a separate directory called bars. This directory became the main training set for the project.

In the create paragraphs function a file is opened an then looped through to find where each paragraph starts and finishes. On lines 7 through 12 the software looks for blank lines and if the lines are blank and the newest paragraph being written contains a line already then the newest paragraph is added to a list and a new paragraph begins.

```
1  def create_paragraphs(filename):
2    my_file = open(filename, 'r')
3    paragraphs = []
4    new_paragraph = []
5    for line in my_file.readlines():
6      if valid_line(line):
7        line = line.strip()
8        if line:
9          new_paragraph.append(line)
10       elif new_paragraph:
11         paragraphs.append(new_paragraph)
```

```
12            new_paragraph = []
13    return paragraphs
```

In the refine paragraphs function code is used to find all of the paragraphs which are written in a four bar structure break them up into their individual bars and add them to a list which is then returned. On line 5 there is a check to see if the paragraph is a multiple of four lines indicating it contains bars. In lines 11 through 15 these paragraphs are then spit into individual bars.

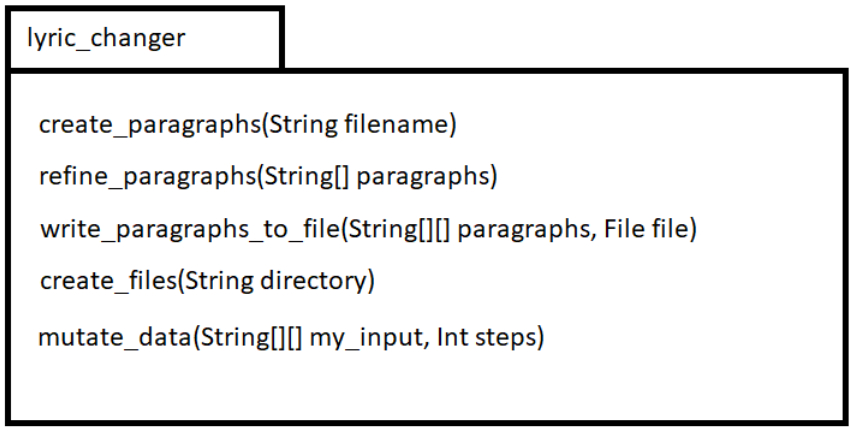
```
1  def refine_paragraphs(paragraphs):
2    bar_paragraphs = []
3    new_paragraph = []
4    for paragraph in paragraphs:
5      if len(paragraph) % 4 == 0:
6        large_paragraph = []
7        for line in paragraph:
8          large_paragraph.append(line)
9        number_of_bars = len(large_paragraph)/4
10       for i in range(number_of_bars):
11         new_paragraph.append(large_paragraph[0 + i])
12         new_paragraph.append(large_paragraph[1 + i])
13         new_paragraph.append(large_paragraph[2 + i])
14         new_paragraph.append(large_paragraph[3 + i])
15         bar_paragraphs.append(new_paragraph)
16         new_paragraph = []
17   return bar_paragraphs
```

The function mutate data is used to shuffle the lyrics. This means that bars by different artists are mixed together which stops artists who appeared later in the training set from having larger influence. The number of paragraphs is obtained on line 2. Two random numbers are then produced on lines 4 and 5. If these numbers are not the same then the paragraphs that these numbers correspond to are swapped on lines 7, 8 and 9.

```
1  def mutate_data(my_input, steps):
2    input_range = len(my_input) - 1
3    for i in range(steps):
4      first_index = randint(0, input_range)
5      second_index = randint(0, input_range)
```

```
6         if first_index != second_index:
7           temp = my_input[first_index]
8           my_input[first_index] = my_input[second_index]
9           my_input[second_index] = temp
10      return my_input
```

This software is very specialised and is unlikely to ever see use outside of this specific use case and project. As such less time was spent on planning and architecture.



3.1.4 Rap Battler

The Rap Battler was create to provide an interface for the user to interact with the neural network. Rap Battlers main responsibilities are to initiate the network, accept input in the form of rap lines, feed them into the network, modify the networks output and to deliver the output to the user. It continues this process for a set period of time until the rap battle has concluded. The main purpose of this system is to provide a more ergonomic and approachable way for a user to play or interact with the Neural Network.

The Rap Battler was built for Python 2.6+ as it needed to interact with the Neural Network which was also created in Python 2.6+. Rap Battler uses the

tensorflow library to load and reconstruct a previously trained Neural Network. It then request input from the user in the form of two lines of text. These two lines of text are then fed into the Neural Network as the seed data. Rap Battler then cuts the output of the network to two lines. These two lines are then passed to a text to speech engine name Pyttsx. This library reads the two lines out to the user which are also printed to screen. The engine uses the Scotch English preset for a more pronounced delivery of lines. At this point the system requests another two lines of dialogue from the user. This process is repeated several times in order to create a 'rap battle' type scenario.

On the first and second lines the software reads the configuration of the neural network in and then on the third line the configuration is used to reconstruct the trained network. On line 8 the user input is accepted and on line 9 that input is passed on to a function that returns the networks formatted response. On lines 13 and 14 the output of the network is passed to a text to speech engine to be read aloud.

```
1  with open(os.path.join(save_dir, 'config.pkl'), 'rb') as f:
2    saved_args = cPickle.load(f)
3    model = Model(saved_args, training=False)
4
5  print('Lay some bars... \n')
6
7  for i in range(args.turns):
8    input = get_input()
9    raw_text = battle(input, args.sample, model)
10   output = format_text(raw_text)
11   for i in range(2):
12     print(output[i])
13     engine.say(output[i])
14     engine.runAndWait()
```

In the battle function the neural network is called upon to give a response to user input. On line 2 the response length is set at 2000 characters. This is enough characters that the network will always produce at least 2 lines of response text. On lines 4 and 5 the vocabulary and characters of the network are loaded up. On line 12 these variables along with the user input and the

current tensorflow session are passed into the network to obtain a response. The response is then returned back up at line 13.

```
1  def battle(prime, sample, model):
2    char_pool_size = 2000
3    save_dir = 'save'
4    with open(os.path.join(save_dir, 'chars_vocab.pkl'), 'rb') as
     f:
5      chars, vocab = cPickle.load(f)
6    with tf.Session() as sess:
7      tf.global_variables_initializer().run()
8      saver = tf.train.Saver(tf.global_variables())
9      ckpt = tf.train.get_checkpoint_state(save_dir)
10     if ckpt and ckpt.model_checkpoint_path:
11       saver.restore(sess, ckpt.model_checkpoint_path)
12       generated_text = model.sample(sess, chars, vocab,
     char_pool_size, prime, sample).encode('utf-8')
13       return generated_text
```

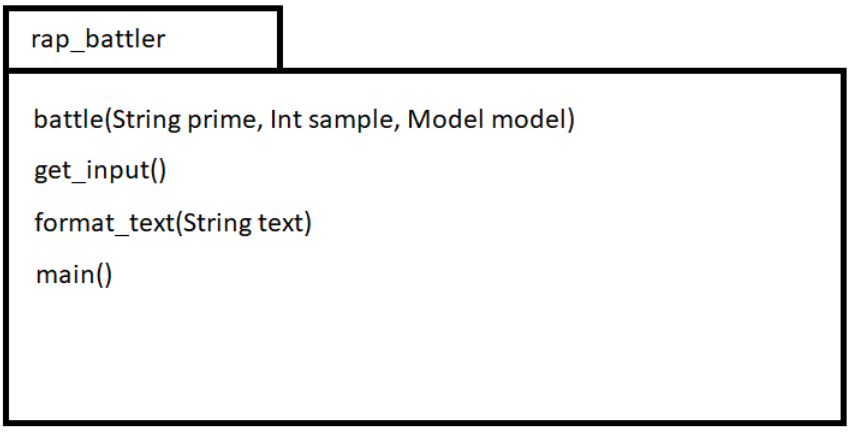
On line 2 the response of the network is split into lines. On line 5 there is check in case the line is blank and then

——————potential bug LOOK INTO ——————

```
1  def format_text(text):
2    lines = text.splitlines()
3    valid_lines = []
4    for line in lines:
5      if line != '':
6        valid_lines.append(line)
7        valid_lines.pop(0)
8    return valid_lines
```

This piece of software is very specialised and was really created with the sole purpose of letting people of a non engineering or research background to interact with the system and body of work created. It contains publicly accessible methods which allow other developers to generate responses to any input or to format text retrieved from different sources independently. Other developers may simple pass a different model into the battle method top receive a

response. This means some of this code may be used in similar research or projects in the future.



This piece of software came with the largest room for improvement. If this project were to be undertaken again in a commercial environment this software could be upgraded to use a web service API for a cleaner more natural human voice. This would greatly enhance the end user experience. Also with the recent widespread propagation of deep learning software used for converting audio to video it would be possible to have a human head speak the lyrics produced by the network.

3.2 The Recurrent Neural Network

For synthesising lyrics a recurrent neural network was chosen. These reasons were:

1. Recurrent Neural Networks can be used for generating lyrics on a character by character level. This means that they are better suited to noticing and developing assonance. Potash et al. (2015)

2. There high variability in terms of selecting training parameters meant that there is a high level of granularity in training measures. This allows for a high level of configurability.

3. There were already existing libraries such as tensorflow which allowed for quick prototyping and a shorter ramp up time to experimentation.

In the area of Recurrent Neural Networks the derivative best suited to NLP is the Long Short-Term Memory model. This model adds an extra layer of complexity to the individual neurons such that they may now remember or forgot information. This allows for longer term dependencies Hochreiter and Schmidhuber (1997) meaning a more consistent and coherent body of work. It also gives way to more complex structure which may play out over several words or lines Potash et al. (2015). The model used in this project was an adapted version of one used in the Ghostwriter paper Potash et al. (2015). This model was built in Python using the Tensorflow library for almost all machine learning related tasks.

3.2.1 Training the Model

Before the model may produce any lyrics it first needs to be trained on an artist. There are many parameters which need to be carefully taught about before training the model. These are as follows:

- **Artist** is the most obvious parameter that must be chosen. The artist should exhibit many essential traits such as a consistent body of text so

that the model has a style to converge on. They must have a rhyming scheme that may be imitated. The size of their vocabulary must also be taken into account. An artist with a larger vocabulary will be harder to generalise. It is also important to avoid artist whom have collaborated a lot in the past as their training set will be polluted with their training set will be polluted by other artist's styles.

- **Number of Neurons** this dictates the size of the hidden state. This defines how long it will take to train the network and it's ability to generalise.

- **Number of Layers** which will effect the networks ability to make long term connections and to generalise better. A higher number of layers will also add to the training time required.

- **Batch size** is the number of inputs which will be summed and averaged before making any adjustments to the weights of the neurons.

- **Number of Epochs** is the amount of times that the training set will be fed into the model. This means that if the number of epochs is set to 4 then every item in the training set will be inputted 4 times.

- **Learning rate** will dictate how likely the network is to hold onto old patterns it has adopted. A high learning rate means that things may be learned quickly but also forgotten quickly.

- **Gradient Clipping** is a measure taken to diminish the exploding gradient problem Pascanu et al. (2012). It clips all values to be below a certain number.

- **Weight Decay** is a technique used to eliminate weights that are rarely reinforced. Any weight that is never increased over a long period of time will slowly decay towards zero.

These values may be changed and manipulated for empirical studies in order to find the appropriate arrangement for lyric synthesis.

3.2.2 Outputting Lyrics for Experiments

These model I have chosen is a character level Recurrent Neural Network. This means that the network is producing the text one letter at a time. This means there will be many different layers of decisions going on at any point.

The first decision that the network must make is at the character level. It must look at the previous character and decide what character are and aren't valid. For example the letter 'q' must always be followed by a 'u' or more clearly still a lower case letter must be followed by another lower case letter or a space. Once the network has decided what characters it may type it moves up another level in the temporal coupling tree.

Now the model must look at the potential words that may be spelled. For example, it may be valid to follow the letter 'h' with a 'y' but in the context of the string 'th' to continue with a 'y' would make very little sense as it would not be possible to spell a word from that position. This means that the network must now keep track of possible words it can spell at any instance. Once it has a set of words it may spell it goes up another level in temporal coupling.

Once the model knows what words it may spell it must look at the previous words in the sentence and see which possible words are grammatically correct. This means that the network cannot follow a verb with a verb or follow a singular pronoun with a plural verb.

Now that the network has scoped out what characters it may use, followed by what words it may spell that make sense, it now must reach back even further in the sequence and look for things like rhyming scheme. This involves looking at the previous sentences and seeing what words it must attempt to rhyme with.

After this is done the network must look further back again and start making even more complex decisions about structure. These are questions like, is it time to start a new verse, is it time to change to another rhyme scheme, should it revert to a chorus. These are extremely complex decisions that involve a

huge ability to look back through time and to make decisions that factor in many characters which occurred several hundred timesteps previous.

When asked to produce lyrics the network will cycle through this process for a predefined number of characters. Once the process is completed the text will be printed to the screen and a copy of the text will be written to a file for further analysis.

Chapter 4

Empirical studies

4.1 Introduction

In this chapter I discuss the process used to carry out, evaluate and report on the empirical studies. Through these empirical studies I investigate the effectiveness of the Recurrent Neural Network and the Long-Short Term Memory model. This chapter looks into the agency of different parameters and training sets when training a Recurrent Neural Network. I show an optimal set of parameters and training data for generating Hip Hop lyrics.

4.2 Metrics for Evaluation

In these empirical studies the Network's, once trained, were used to produce a large body of lyrics ranging from 10,000 to 100,000 characters in length. These bodies of lyrics were then evaluated using a 5 metrics.

1. **Spelling Error:** is the most fundamental metric used in evaluating the Network's output. Spelling error is the percentage of words in the body of lyrics spelt incorrectly.

2. **Plagiarism:** is another major metric used throughout the empirical studies process. For measuring plagiarism I took every line from the

outputted text and checked if said line could be found anywhere in the training set. I decided to only compare against lines in the training set as lines that may have registered as plagiarised from other sources did not matter as the presence of those lines could only happen through coincidence.

3. **Rhyme Density:** is a measure of the level of assonance found in the text. Assonance is the measure of recurring vowel sounds found in a piece of English writing. It is one of the simpler forms of rhyme to quantify. To attain this metric I used a program called Raplysaattori created for the paper Dopelearning Malmi et al. (2015). This metric is delivered in terms of occurrence of assonance per line of text.

4. **Sentiment:** is a metric I devised myself and is an attempt to capture the sentimental consistency of a body of text. It was created by taking the happy sentiment or sad sentiment of each line which may be given as a value between 0 and 1 and then taking the difference between each subsequent line. The difference is then multiplied by a second value known as polarisation. This value indicates how much sentiment is shown in a line. I take this value produced and get the average between every line of text.

   (a) σ a value tracking the average emotional distance between lines (lower is better)

   (b) L is the line count of the output being tested

   (c) n is the line of text being analysed

   (d) f(n) is the sentiment of the line being analysed

   (e) g(n) is the polarity of the line being analysed

$$\sum_{n=2}^{L-1} \frac{g(n) * ||f(n) - f(n-1)| + |f(n) - f(n+1)||}{L-2} = \sigma$$

5. **Generalization:** is a measure I also created to try and quantify the overall ability of the network to produce both valid and original lyrics as

that is the overall goal of this paper. This meant that there was a need to measure the networks fitness that took into account both the networks ability to write lyrics and its ability to do so without plagiarising. The metric would need to account for both the spelling error of the output and the originality of it too. This metric took into account both the plagiarism rate and spelling error rate of the network.

(a) ω is a normalised value indicating the quality of output

(b) α is a constant weight applied to the spelling error rate

(c) β is a constant weight applied to the originality of output

$$\omega = 1 - \frac{(\alpha * e) + (\beta * p)}{\alpha + \beta}$$

4.3 Characterising Data

An important influence in the empirical studies was that of the training data. In these studies there was ? training sets used. Three of these were were just large plain text files containing an artists corpus. The artists used in this format were Eminem, Lil Wayne and Aesop Rock. The Eminem training set contained ??? lines of text whilst Lil Wayne and Aesop Rock contained ??? and ??? lines of text respectively. It was important to keep in mind the spelling error rate and vocabulary size of these training sets as it had an effect on the effectiveness of any network trained on this data.

Artists Name	Spelling Error	Unique Words	Size
Lil Wayne	11.86%	84.25%	???
Eminem	13.69%	20.82%	???
Aesop Rock	8.09%	70.30%	???
Multiple Artists	12.21%	3.60%	???
Bars	8.62%	5.80%	???

4.4 Template Used

For each experiment there is a specific template used. This template provides the relevant information on each empirical study so as to allow any reader to recreate the results. In each study I briefly outline the following:

The **Objective** of the current study. What this study is designed to discover or disprove. This comes with a small description of what the effect outcome of the experiment should be.

The **Architecture** being used in the current study. This is a brief description of the Neural Network used.

Training Sets are the data sets being used to train the network in the current experiment. This will come with a brief description of what artists were used and important meta data about the training set.

Parameters used when training the model. These parameters are used to configure the model in terms of structure and constants being applied whilst training. There were several parameters that were never changed over the course of the experiments. The batch size and sequence length were fixed at 500 each. Gradient clipping was fixed at 5 to help contain the exploding gradient problem. The learning rate and decay rate of the network were set at 0.002 and 0.97 respectively. All other parameters were altered throughout the empirical studies in order to gain insight into their agency.

Time Taken is the total time that the network was allowed to train for. This is important as Networks being run for the same number of Epochs may vary in runtime depending on a wide range of features.

Metrics are a predefined set of tests and evaluations used to measure the quality of the lyrics outputted by the network and therefore the quality of the network.

The **Results** of each study will be displayed in each section. These will be the output of any metrics applied to the output of the network.

The**Evaluation** section will contain a brief discussion of the results obtained in the study with an exploration of what the results might mean going forward.

4.5 Studies

4.5.1 Study 1: Epochs

Objective

The objective of this study was to gain insight into the influence of epochs when training a neural network. Epochs are defined as "iterations through the training set"Bengio (2012). This means that one epoch is one full pass over the training set.

Architecture

The architecture used was that of the Recurrent Neural Network. This Recurrent Neural Network was built using the Long-Short Term Memory Model.

Training Sets

There was only one training set used in this experiment and that was the Lil Wayne corpus.

Parameters

The network was parametrised with 1 layer and 128 neurons in said layer. The number of epochs used to train the network was increased throughout the study. The network output was set at 100000 characters.

Time Taken

The time to run each experiment was directly proportional to the number of epochs the network was trained for.

Metrics

The metrics applied to these outputs were spelling error and rhyme density.

Results

Table 4.1 shows the number of epochs the network was trained for and the metrics applied to the output of said network once trained.

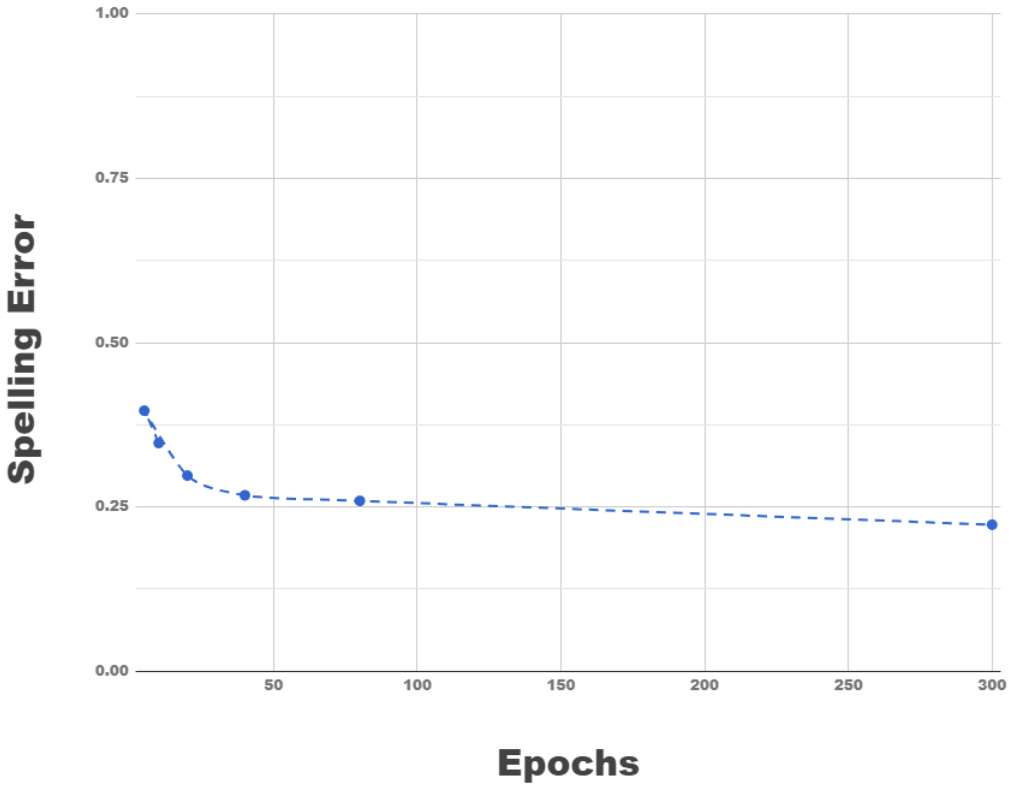
Evaluation

It is clear that Epochs and Spelling Error follows an exponential curve in figure 4.1. Meaning that increasing the epochs a network is trained upon produces diminishing returns as spelling error bottoms out.

Table 4.1: Effect of Epochs on Spelling Error and Rhyme Density

Epochs	Spelling Error	Rhyme Density
5	39.64%	1.030
10	34.93%	0.989
20	29.76%	0.994
40	26.76%	1.006
80	25.91%	1.020
300	22.28%	0.989

Figure 4.1: Effect of Epochs on Spelling Error Graphed



4.5.2 Study 2: Network Size

Objective

To investigate the effect of the networks size on the ability of the network to converge upon a solution.

Architecture

This study utilised the same architecture as study one.

Training Sets

The training set used in this study was the same training set as used in study one.

Parameters

All parameters were fixed in this study except for epochs, number of layers and the number of neurons in each layer. The network output was set at 100000 characters.

Time Taken

The time taken for the first experiment using 128 neurons and one layer was only run for only 4 hours as the network appeared to stop converging after sufficient epochs. The other two experiments used four layers with 256 and 512 neurons per layer respectively and were run for 13 hours each.

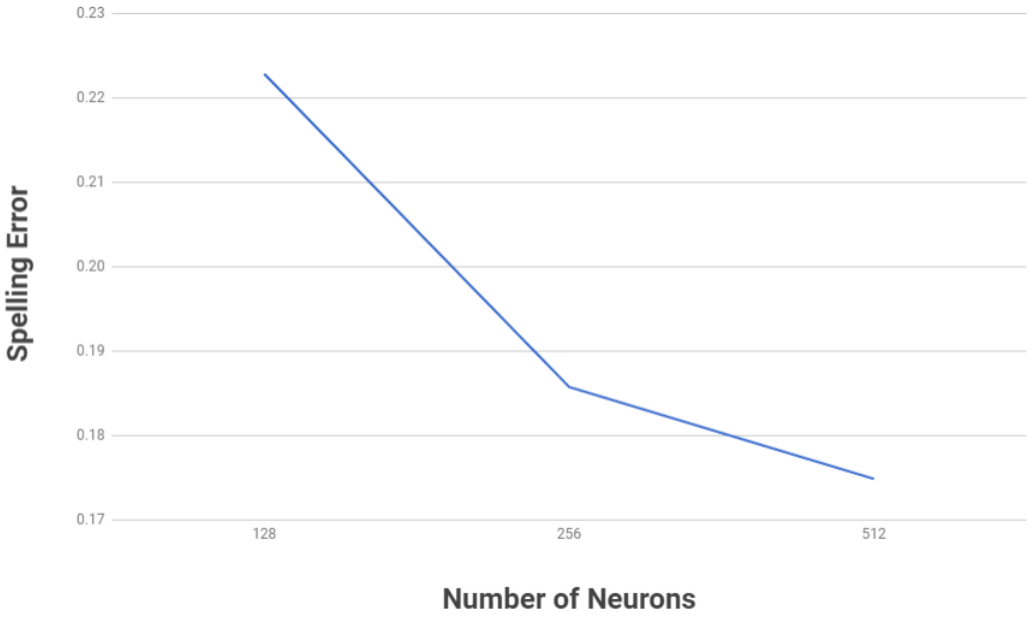
Metrics

The metrics used in this experiment were spelling error and rhyme density.

Table 4.2: Effect of Neurons per Layer on Spelling Error and Rhyme Density

Epochs	Spelling Error	Rhyme Density
128	22.28%	0.989
256	18.57%	0.971
512	17.49%	1.023

Figure 4.2: Effect of Neurons on Spelling Error Graphed



Results

Evaluation

It is clear that the addition of more neurons to the network has improved the networks ability to spell words.

4.5.3 Study 3: Training Sets

Objective

To investigate the effect of different training sets on the ability of the network to converge on solution that produces quality lyrics.

Architecture

This study uses the same architecture as study one.

Training Sets

The training sets were varied throughout this experiment in order to assess their effect on the training of the model. The training sets used in this project were the corpus's of Lil Wayne, Eminem, Multiple Artists and Aesop Rock along with the bars data set.

Parameters

The network was parametrised with 512 neurons in a layer and four layers. The number of epochs was adjusted per training set so as to have each network trained for a similar amount of time. The two experiments that used Lil Wayne and Aesop Rock was terminated earlier than expected as the network had seemed to stop improving. This could be due to Lil Wayne's small vocabulary and the relatively small size of the Aesop Rock training set.

Table 4.3: Effect of Training Sets on Spelling Error and Rhyme Density

Training Set	Spelling Error	Rhyme Density
Lil Wayne	11.86%	1.003
Aesop Rock	8.09%	0.995
Eminem	13.69%	1.000
Multiple Artists	12.21%	0.945
Bars	8.62%	1.011

Figure 4.3: Results Achieved When Trained on Differing Training Sets

Time Taken

Each model was trained for 160 hours with the exception of the Lil Wayne and Aesop Rock trained networks which were trained for 120 hours.

Metrics

The metrics used in this experiment were spelling error and rhyme density.

Results

Evaluation

Figure 4.3 shows that the training set used to train the network can have an effect on the quality of output produced by the network. According to table 4.3 a network trained on Aesop Rock makes less than 60% as many spelling mistakes as a network trained on Eminem's corpus.

4.6 Studies Revisited with Generalization

4.6.1 Study 1: Epochs

Objective

Architecture

Training Sets

Parameters

Time Taken

Metrics

Results

Evaluation

4.6.2 Study 2: Network Size

Objective

Architecture

Training Sets

Parameters

Time Taken

Metrics

Results

Evaluation

4.6.3 Study 3: Training Sets

Objective

Architecture

Training Sets

Parameters

Time Taken

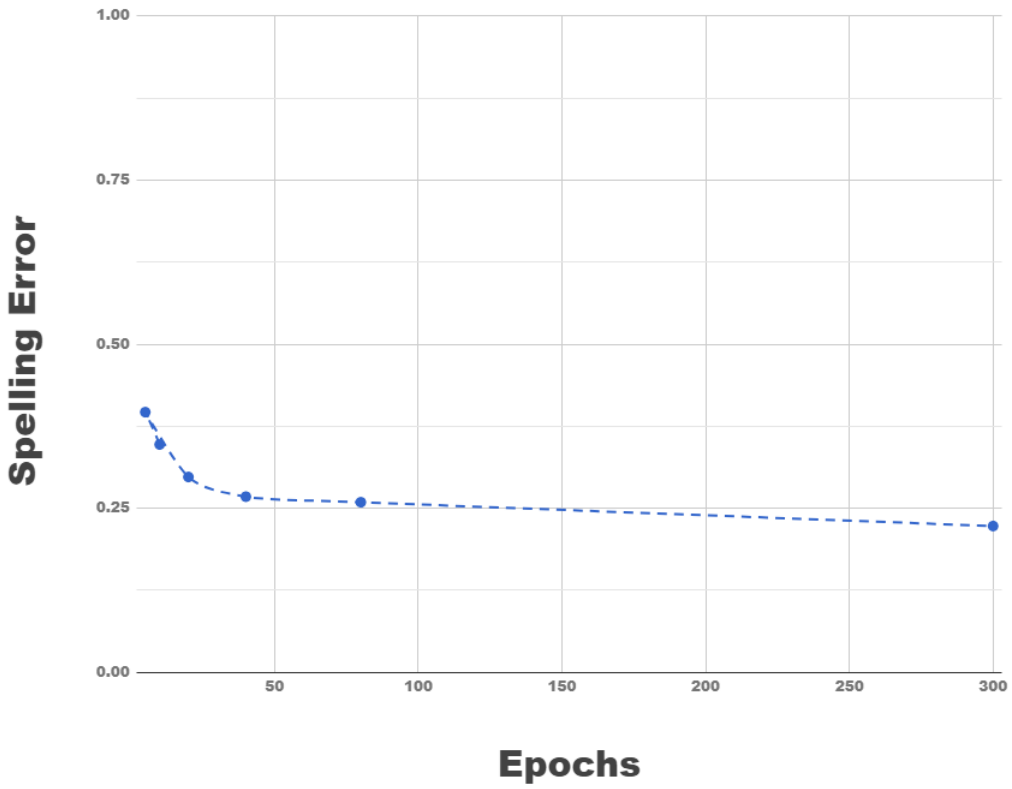
insight into the optimal set of parameters when training a Recurrent Neural Network to produce lyrics.

4.8 128 Neurons 1 Layer

In the beginning a small network of just 128 neurons and one layer was used. The original goal was to push this network as far as possible but it became apparent as soon as the epochs were increased the network hit a point of diminishing returns. The network struggled to learn a large vocabulary of words and despite only 25 percent of the words showing up as spelt wrong this was largely due to the network sticking to smaller words.

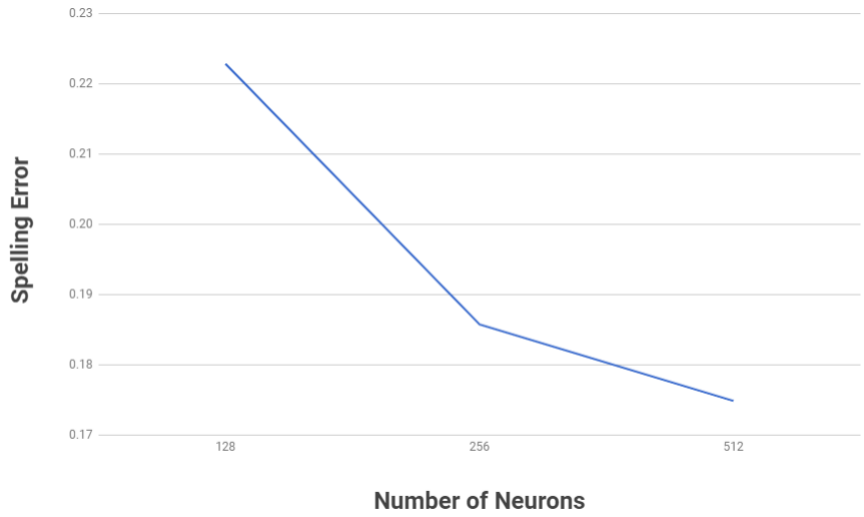
"When I don't you'll just at me like 'em from hard boHy I shines and shit I'm a gloeney gone take me wanna Help. Swipsin', drop with twe said she learn ya'll flowers Money pake it ass Weezy So I ain't see me, tell ceepin' that Been poul out are my nutchin' five 4 dumb to my funass"

Only a single word over 5 characters is attempted and spelt correctly in this extract and little to no grammer can be found. This shows that the small network is incapable of learning to rap.



4.9 Increasing Network size

This made it obvious that there was a need to increase the overall size of the network. It was shown that as the number of neurons was increased the quality of the output began to rise too until also seeing a point of diminishing returns but at least this was progress.



At this point it was worth noting the use of Lil Wayne in this set of experiments. The reasoning behind using this artists corpus was the artists small vocabulary and large body of work. With a spelling error rate of 11 percent in his work it became apparent that the network was approaching his quality of work.

4.10 Training Data Quality

Attention was once again redirected in the pursuit of better results. The use of different artists works and their influence on the quality of output produced by a network trained on said work. A small selection of artists was assembled based upon vocabulary size, grammer and size of available training data.

A network with 4 layers and 512 neurons per layer was then trained on the data set of Aesop rock, Eminem and a conglomeration of several artists.

4.11 Generalization

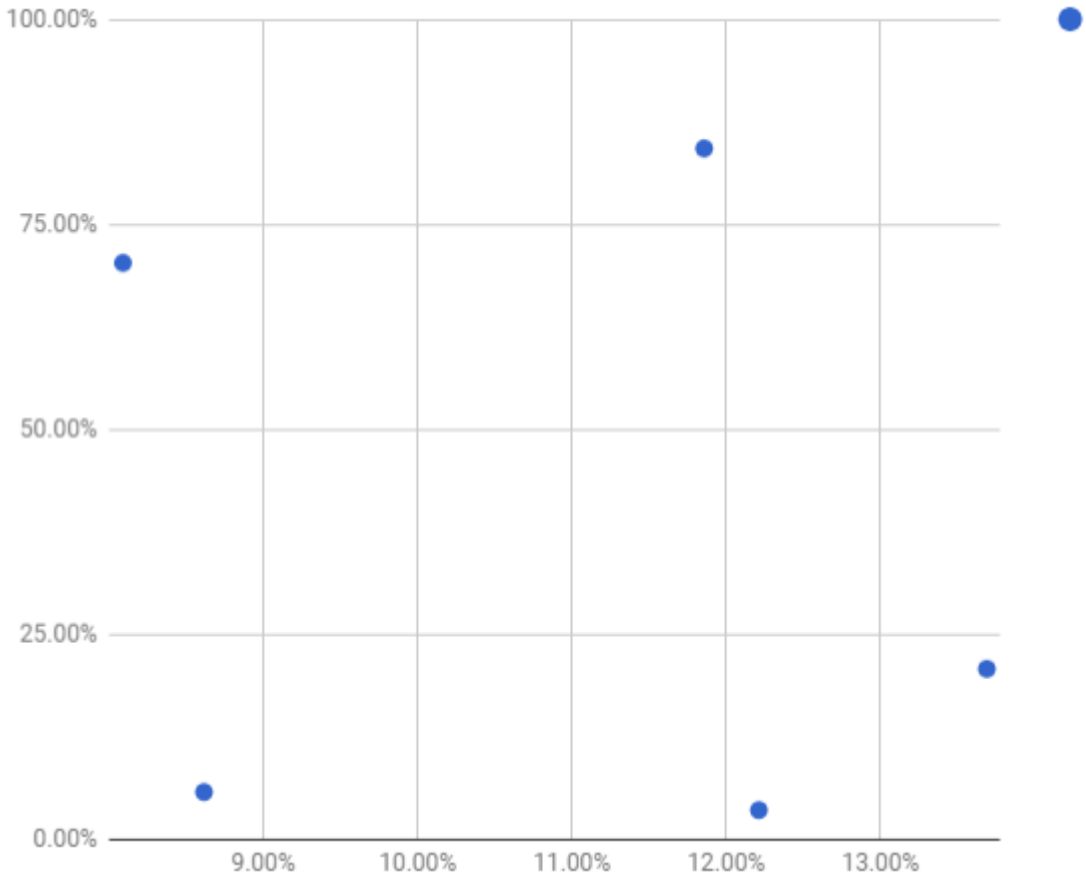
The network now ran into a new problem in that it would over fit the training data. In the case of lyric generation that meant that the model copied lines and even verses from the artists corpus wholesale. This would mean that the model had failed to generalize. That meant that in the use case of a rap battle the model would struggle to accommodate words or phrases it had never seen before. Here is an example using Aesop Rock. The seed data here was "It was all a dream".

" It was all a dreams only the kitched to space (aliary waster, what's up. Well I'vey see his zog' bug they hands out "

Whilst the output for experiments measured an 8 percent spelling error the lines proceeding the input line were far from that standard.

Artists Name	Spelling Error	Plagiarism
Lil Wayne	11.86%	84.25%
Eminem	13.69%	20.82%
Aesop Rock	8.09%	70.30%
Multiple Artists	12.21%	3.60%
Bars	8.62%	5.80%

Spelling error was plotted against plagiarism in order to see if there was any correlation between the pair.



I began by giving α and β the same value. In this case I gave them both a value of 1.

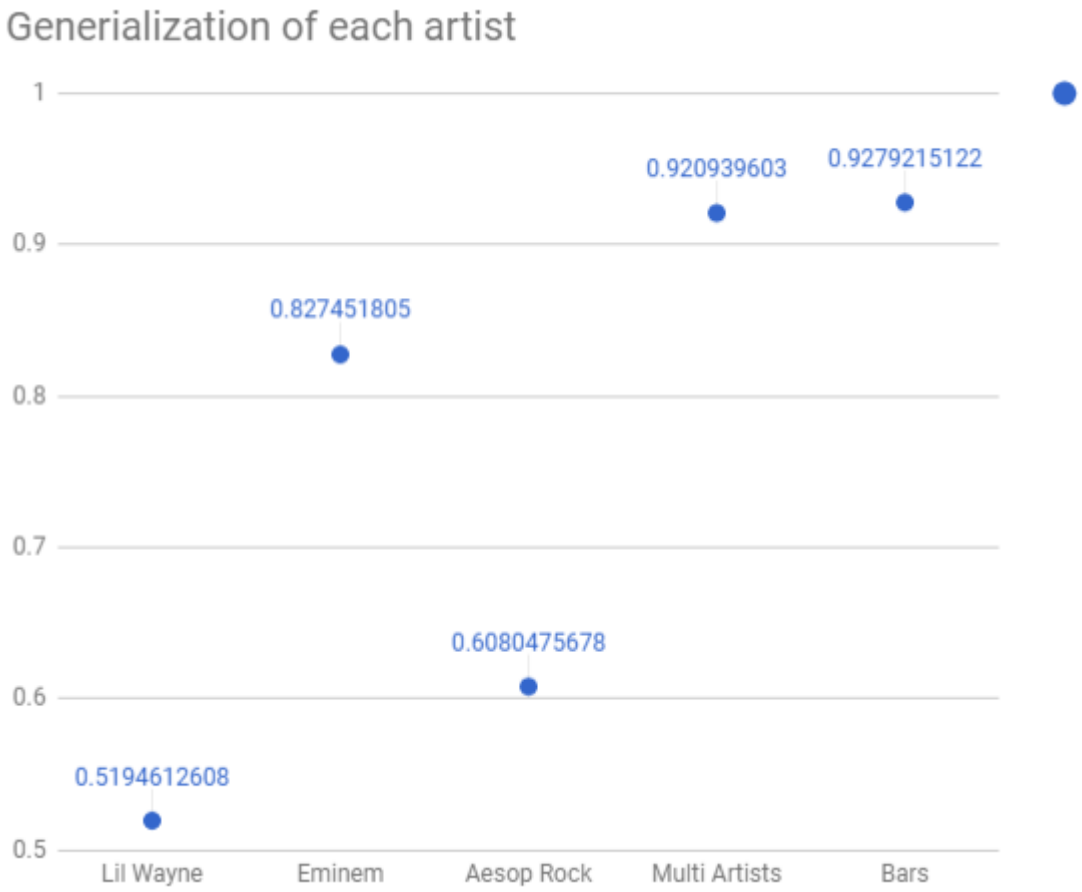
This gave me a formula that looked like this when evaluating Lil Wayne;

$$\omega = 1 - \frac{(1 * 0.1186) + (1 * 0.8425)}{1 + 1}$$

This gave Lil Wayne a normalised value of 0.5194612608.

This process was then repeated for each training set and their corresponding output.

Artists Name	Spelling Error	Plagiarism	Generalization
Lil Wayne	11.86%	84.25%	0.5194612608
Eminem	13.69%	20.82%	0.827451805
Aesop Rock	8.09%	70.30%	0.6080475678
Multiple Artists	12.21%	3.60%	0.920939603
Bars	8.62%	5.80%	0.9279215122



4.12 Sentiment

Bibliography

Gobinda G Chowdhury. Natural language processing. *Annual review of information science and technology*, 37(1):51–89, 2003.

D. Sarkar. *Text Analytics with Python: A Practical Real-World Approach to Gaining Actionable Insights from your Data.* Apress, 2016. ISBN 9781484223888. URL `https://books.google.ie/books?id=IimgDQAAQBAJ`.

Gabriele Barbieri, François Pachet, Pierre Roy, and Mirko Degli Esposti. Markov constraints for generating lyrics with style. In *Proceedings of the 20th European Conference on Artificial Intelligence*, pages 115–120. IOS Press, 2012.

Eric Malmi, Pyry Takala, Hannu Toivonen, Tapani Raiko, and Aristides Gionis. Dopelearning: A computational approach to rap lyrics generation. *arXiv preprint arXiv:1505.04771*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2012.

Peter Potash, Alexey Romanov, and Anna Rumshisky. Ghostwriter: using an lstm for automatic rap lyric generation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1919–1924, 2015.

Andres Munoz. Machine learning and optimization. *URL: https://www. cims. nyu. edu/˜ munoz/files/ml_ optimization. pdf [accessed 2016-03-02][WebCite Cache ID 6fiLfZvnG]*, 2014.

David E Wilkins. *Practical planning: extending the classical AI planning paradigm*. Morgan Kaufmann, 2014.

KJ Singh. What are the limitations of the expert systems?, 2010. URL `http://www.mbaofficial.com/mba-courses/principles-of-management/what-are-the-limitations-of-the-expert-systems/`.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

Marjan Ghazvininejad, Xing Shi, Yejin Choi, and Kevin Knight. Generating topical poetry. In *EMNLP*, pages 1183–1191, 2016.

Dongzhuo Li, Chao Liang, and Tianze Liu. Automatic generation of lyrics in bob dylan's style.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Vincent Granville. Training a computer to recognize your handwriting, 2016. URL `https://www.datasciencecentral.com/m/blogpost?id=6448529%3ABlogPost%3A408676`.

Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, STANFORD UNIV CA STANFORD ELECTRONICS LABS, 1960.

Marvin Minsky and Seymour Papert. Perceptrons. 1969.

J-P Dodel and Rajjan Shinghal. Symbolic/neural recognition of cursive amounts on bank cheques. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 15–18. IEEE, 1995.

Tapan Sau. Breath acetone-based non-invasive detection of blood glucose levels, 06 2015.

Geoffrey E Hinton. How neural networks learn from experience. *Scientific American*, 267(3):144–151, 1992.

Richard Lippmann. An introduction to computing with neural nets. *IEEE Assp magazine*, 4(2):4–22, 1987.

Lipo Wang. *Support vector machines: theory and applications*, volume 177. Springer Science & Business Media, 2005.

S. Abe. *Support Vector Machines for Pattern Classification*. Advances in Computer Vision and Pattern Recognition. Springer London, 2006. ISBN 9781846282195. URL `https://books.google.ie/books?id=3DswtC\_ZYrwC`.

Patrick Winston. 16. learning: Support vector machines, 2014. URL `https://www.youtube.com/watch?v=_PwhiWxHK8o&t=2023s`.

Dan Jurafsky and James H Martin. *Speech and language processing*, volume 3. Pearson London, 2014.

Lyrics freak. URL `www.lyricsfreak.com`.

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer, 2012.