Mathematical Approaches to Machine Translation

Rhys T Nicholas

Division of Mathematics and Statistics

University of South Wales

A submission presented in

partial fulfilment of the requirements

of the University of South Wales/Prifysgol De Cymru

for the degree of BSc Mathematics

April 1, 2020

UNIVERSITY OF SOUTH WALES
PRIFYSGOL DE CYMRU

FACULTY OF COMPUTING, ENGINEERING & SCIENCE
SCHOOL OF COMPUTING & MATHEMATICS

STATEMENT OF ORIGINALITY

This is to certify that, except where specific reference is made, the work described in this project is the result of the investigation carried out by the student, and that neither this project nor any part of it has been presented, or is currently being submitted in candidature for any award other than in part for the degree of BSc or BSc with Honours of the University of South Wales.

Signed: *Rhys Nicholas*

Date : *30/03/2020*

Abstract

This research study focuses on understanding the fundamentals of Machine translation in the construction of simple machine translators. Three machine translation models for English to Welsh have been produced in Python with the final Model yielding a measurable level of real translation. All models have been tested using Word Error Rate and BLEU translation quality measures and compared with Google Translate to show an accurate representation of their translation capacity.

A further aim of outlining areas of difficulty for Machine Translation for a minority language has determined the lack of high-quality data and limited resources as the single most obstructive problem restricting progress.

Contents

Chapter 1

Introduction

1.1 Brief History of Machine Translation

Machine translation (MT) is the process of translation between two languages, typically referred to as Source (SL) and Target languages (TL), through the use of machines and computers. The subject is historically recent having originally been proposed by Warren Weaver in 1949 in a memorandum simply titled "Translation".[1]
Progress in the field was small and overstated culminating until 1966 when the ALPAC report (*Automatic Language Processing Advisory Committee*) condemned the field causing ripples of de-funding and disrupting progress in the field for years to come.[2] [3]

Significant steps as a result of the field were next best seen with the movement of machine translation to the internet with free use online translation tools, such as with SYSTRAN and AltaVista Babelfish in the 1990s.[4] Currently this is most notably being seen with with Google Translate, originally launched in 2006 as a statistical translator but swapping to Neural Networking in 2016, Google translate currently offers translation between over 100 languages.[5]

Mathematical approaches to the field are predominately seen in the underlying systems of the applied translation models which draw logic from fields such as statistics and information theory. Further comparisons can be made with fields such as cryptography which has extensive study in the properties of language as well as comparisons to the computational effort to optimise machine translation systems for efficiency.

This project aims to approach the high level field of MT and to understand the fundamental concepts in order to apply them in the construction of simple machine translators. An additional underlying goal is to outline the areas of difficulty for building MT systems for a minority language with limited resources.

1.2 Types of Machine Translation

Machine translation can be categorised into four main types: Rule-based Machine Translation (RBMT), Statistical Machine Translation (SMT), Hybrid Systems and Neural Networks.[6]

Rule Based MT (RBMT)

These systems translate based on sets of linguistically predetermined rules designed specifically for the SL and TL. Typical rules could be dictionaries for both languages as well as grammatical rules for tense, structures and word order. The more different two languages are the more rules necessary to produce a reasonable level of translation.
The underlying principle of RBMT is to create a connection of the underlying meaning of the SL input with a TL output.

Satistical MT (SMT)

Statistical MT draws logic from information theory and statistics with applications such as Bayes Theorem for n-gram size words or sentences. n-grams are defined as some sequence of n items, typically words or letters where n denotes the number in the sequence. For example, a word of length n can be split into letters, 1-gram letters, or pairs of letters, 2-gram letters, or taken as a whole either as n-gram letters or as a 1-gram word.

The general idea behind SMT can be explained by first considering some n-gram sentence in the SL, x for which there exists various corresponding translations in the TL. Let y denote some n-gram string in the TL, for every possible pair of (x, y) there is an associated probability that y would be given as a translation for x, $P(y|x)$ which by Bayes theorem can be given as,

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

Further still, consider that a given TL string y has some x of greatest probability given by $\hat{x}$ and assume that the n-gram y is in fact the translational equivalent of $\hat{x}$ to a native speaker of the target language. This optimisation is then given by,

$$\hat{x} = \text{argmax}\{P(x|y)\} = \text{argmax}\left\{\frac{P(y|x)P(x)}{P(y)}\right\} = \text{argmax}\left\{P(y|x)P(x)\right\}$$

Which comes from the fact that $P(y)$ is independent of x and so it is only necessary to maximise the product $P(y|x)P(x)$. This is a generalisation of the wider statistical models applied in modern systems, but functions as a representation of the underlying logic.[7]

Hybrid MT

Hybrid Machine Translation Models are the combination of RMBT with SMT in ways to optimise the translation by minimising the areas the systems respectively fail. The ways in which they are combined vary, the multi-pass method involves passing a translation through multiple sub systems, typically a RBMT generated translation through a statistical proofer in order to reduce the amount of statistical and lexical rules necessary.
The multi engine works by running statistical and RBMT systems in parallel and combining the results for an optimised translation whereas the Statistical Rule generator involves using Statistical data to produce lexical rules which are then applied as normal for a rule-based system.[8]

Neural Network MT

Neural network systems can be trained to translate between a given pair of SL and TL. This is done by feeding the system parallel texts translated down to the sentence. These are treated as end to end pairs where the SL text is the input end and the TL is the output end. With enough data and iterations that optimise for successful translations, the system can begin to give translations. This can be generalised as the Neural Network being trained to spot patterns in the data in order to produce similar results.
As the systems are constructing themselves by iteration a level of reverse engineering is necessary with this approach to ensure the patterns being made are relevant to the core aim of the system.[9]

1.3 Languages

This project focuses on the translation between the languages English and Welsh, with English as the source language, and may use the abbreviations En and Cy respectively. Although English and Welsh are from bordering nations with an interconnected cultural history the languages are lexically and grammatically very different originating from different language families, Germanic and Celtic.

En : ***The cat ate the mouse*** = *Article, Noun, Verb, Article, Noun*

Cy : ***Bwytodd y gath y llygoden*** = *Verb, Article, Noun, Article, Noun*

Examples of the grammatical differences range from areas such as Word order, where Welsh uses Verb Subject Object but English uses Subject Verb Object, as seen in the above example. To alphabetic where Welsh does not possess the letters **Z, Q, K, J, V** but does have 2-gram letters that are defined as individual letters **Ch, Dd, Ff, Ng, Ll, Ph, Rh, Th**. Further differences include differing phonetics as well as differing grammatical constructions.[10][11]

1.4 Lexical Similarity

Lexical Similarity is defined as the distance between languages and can be thought of as a measure that determines how lexically close two languages are. Various methods exist and approaches can range from Morphological focusing on the words and their meanings to Phonetic focusing on the underlying sounds.[12]

1.4.1 Root Mean Square Error

As no formal measure of similarity has been produced for Welsh and English this project has investigated a novel application of the Root Mean Square Error (RMSE), for which the equation is given below.[13]

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum(X_c - X_o)^2}$$

The RMSE for a set of Calculated Data X_c and a set of Observed Data X_o gives a measure of error between the Data sets. The equation has roots in Euclidean distance and can be considered a measure of difference or distance between the sets of data.[14] As a measure for lexical similarity the calculated and observed data X_c and X_o could be any quantifiable regularities of words or letter usages in two languages. This was applied using the frequency of 1-gram and 2-gram letters in English compared with that of Welsh, German (De) and French (Fr).[15][16][17][18]

The data was calculated using frequency lists of words as the source data and ignores letters that are not in English as well as accent marks. The results give a difference rather than a similarity, which has been converted to a measure of similarity by first considering the RMSE of a calculated frequency set X_c of English with an observed set of data X_o all equal to 0. This gives a distance that can be generalised as a furthest from English, let this be $RMSE_{En}$. The Similarity is then given by,

$$S = 1 - \frac{RMSE_{En-C}}{RMSE_{En}}$$

For S the similarity and C the compared language. The results are tabulated below,

	En - Cy	En - De	En - Fr
1-Gram	48.18%	63.94%	66.19%
2-Gram	6.04%	82.46%	82.04%

Although a simplification of the morphological part of language, the results show a higher similarity for English with German and French than Welsh, especially so for 2-grams, despite the geographical closeness of the languages.

Chapter 2

Model 1: Dictionary Method

2.1 Method

One of the simplest methods to translate text from one language to another is to replace each word of the SL with an equivalent in the TL, a word for word method here named the dictionary translator method. The first public demonstration of MT in 1954 was a dictionary translator, produced by a collaboration between IBM and Georgetown University, that attempted Russian to English translation. Although this inspired a growth in MT during the late 50s, this system was heavily constrained using a dictionary of 250 words and only 6 rules.[19]

2.2 Assumptions

To produce a model, first it is needed to consider the necessary assumptions in order to produce the system given the level of resources available. These assumptions will also keep the system in question simple which is preferred for any initial model.
The main assumption is that of a one to one relationship between the SL and the TL, such that for any given word in one language there exists a single equivalent word in the other. This is untrue in general for languages, as words can hold different meanings in different contexts which may be translated as separate words. For example, in English the word ***Free*** can be translated in the two following ways,

$$\textbf{Free} = \begin{cases} \textbf{Am ddim} = \textit{Free of cost} \\ \textbf{Rhydd} = \textit{Available or Unrestricted} \end{cases}$$

A further assumption is that the grammar of the TL is shared by that of the SL, this is a result of the model not considering grammar in the translation. Therefore, any grammar used for the SL will simply be carried through to the TL.

2.3 Constraints

Defining a constraint to be a change to a fundamental aspect of the system, the only constraint made is the substitution of the source language English for Basic English to ensure simplicity during construction.

Basic English (BE), originally developed in 1925 by Charles K Ogden, is a linguistically designed subset of English that uses a reduced dictionary of 850 words along with simplified grammatical systems and rules.[20] Although simple, many of the properties of English are still maintained and expanded word list variants exist.

2.3.1 Zipf's Law

The difference between Basic English and English is well represented through Zipf's law, an Empirical law that applied to linguistics gives the relationship that the frequency of a word in a given language is inversely proportional to its rank. That is, the most frequent word will occur twice as often as the second most frequent and three times as often as the third most frequent, etc.

$$r_n \propto \frac{1}{f_n} \implies r_n = \frac{k}{f_n}$$

r_n gives the rank of a word of n^{th} most frequency in ascending order, such that the most frequent word has a rank of 1. The n^{th} most frequency is given by f_n and k is a constant of proportionality.[21] This relationship is best represented when graphed.

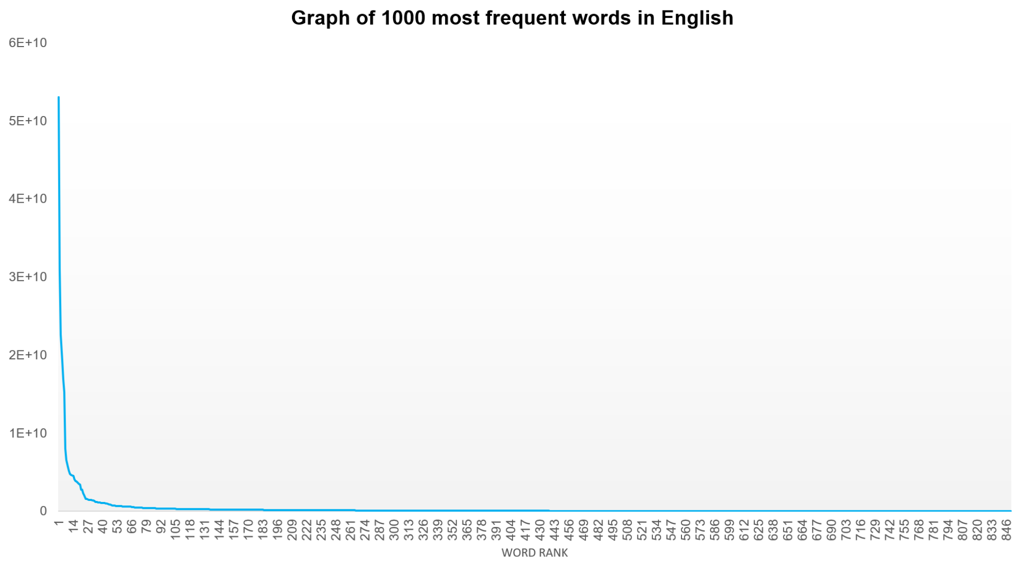


Figure 2.1: Zipfs Curve for the 1000 most frequent English words

Figure 2.1 shows the Zipf's curve for English proper, using the same data source for word frequency as used in any models. The curve is more noticeable for larger sample sizes of Words and shows the underlying logarithmic relationship involved. It can also be seen that a majority of the frequency of words in English is held by a minority of the words.[15]

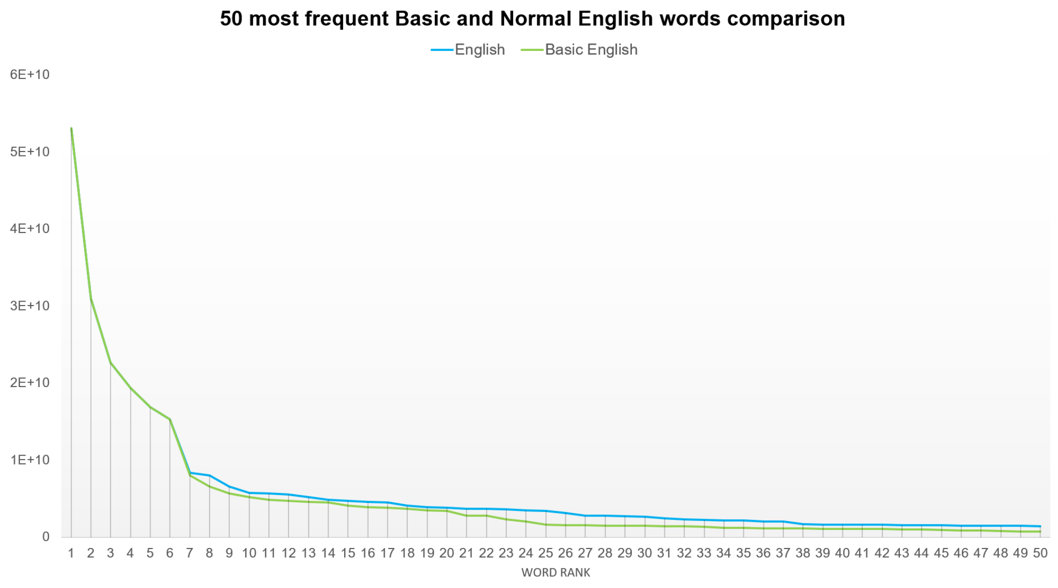


Figure 2.2: Comparison of Zipf's curve for English and Basic English.

Figure 2.2 compares the Zipf's curve of English with Basic English and shows the similarity between them. Every time Basic English omits a word its Zipf's curve decreases faster than English. Considering the total frequency to be the area under the curve, it can be seen that Basic English still holds a large portion of the total frequency. For this data set the frequency of the 850 Basic English words makes up 47% of the frequency of English despite over 97 thousand unique words counted. Therefore, the use of Basic English will still allow for conclusions that apply to English.

2.4 Translation Model

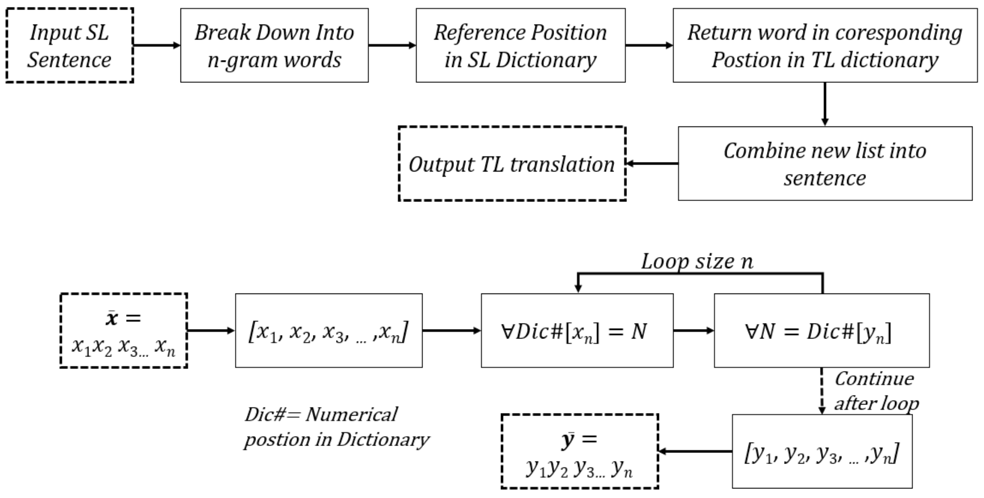
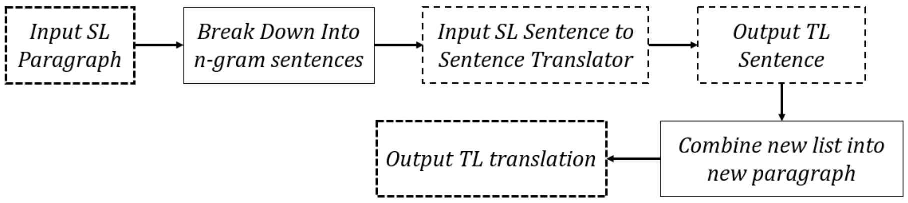

Input SL Sentence
Break Down Into n-gram words
Reference Position in SL Dictionary
Return word in coresponding Postion in TL dictionary
Combine new list into sentence
Output TL translation
Loop size n
$\bar{x} = x_1 x_2 x_3 \ldots x_n$
$[x_1, x_2, x_3, \ldots, x_n]$
$\forall Dic\#[x_n] = N$
$\forall N = Dic\#[y_n]$
Continue after loop
Dic#= Numerical postion in Dictionary
$\bar{y} = y_1 y_2 y_3 \ldots y_n$
$[y_1, y_2, y_3, \ldots, y_n]$

Figure 2.3: Sentence Translator

The python code is split into two functions, the first is shown by *Figure* 2.3, which translates a SL input of sentences without punctuation and references each word individually to a pre-existing list of English words. It then returns the corresponding Welsh word in that position. Any word that was not found in the SL is carried through untranslated.

For this to work a dictionary of Basic English and a one to one corresponding dictionary in Welsh was necessary. To determine the corresponding Welsh words the first translation given from Google Translate was used. This does not mean Google Translate was used in any active part or process of any Models.



Input SL Paragraph
Break Down Into n-gram sentences
Input SL Sentence to Sentence Translator
Output TL Sentence
Combine new list into new paragraph
Output TL translation

Figure 2.4: Paragraph Translator

The second function, *Figure* 2.4, splits up a SL input paragraph into sentences to then be input into the first function to translate them as sentences, returning each result in order. Sentences are defined as strings of words separated by periods. As a result, these periods are the only punctuation carried through to the translation.

2.5 Results

The cat on the floor	$\xrightarrow{\textbf{English}\rightarrow\textbf{Welsh}}$	*y cath ar y llawr*

Figure 2.5: Sentence Translator example input

The sun did not shine. it was too wet to play. so we sat in the house all that cold cold wet day	$\xrightarrow{\textbf{English}\rightarrow\textbf{Welsh}}$	*y haul* ***did*** *nid* ***shine****.* ***it was too*** *gwlyb i chwarae. felly* ***we sat*** *yn y tŷ i gyd bod* ***cold cold*** *gwlyb diwrnod*

Figure 2.6: Paragraph Translator example - excerpt from Cat in the hat[22]

Words that were not translated are in bold, these occurred due to the constraint language Basic English. The choice for translating Cat in the Hat comes from the history of the text being written with a restricted word list whilst still aiming to be simple.

2.6 Review

In terms of the specifications of the model, simple input sentences can yield translation, but complex sentences only yield partial translations. The quality of the translation is poor, being fragmented such that only a bilingual reader can understand the texts.

In terms of translation in general, simple words are not translated, punctuation is limited and as grammar is not considered the grammatical result is distorted and incorrect for Welsh. This has the adverse effect that the meaning of the text is not necessarily being preserved through translation.

2.7 Improvements

The constraint language caused problems for input, expanding this dictionary will allow for both better results as well as results that better represent English. Variations of Basic English exist with modified word pools such as a variant with 1000 words.

This system does not truly translate but does apply relevant logic in the construction of the system. An improved model could use this logic as a foundation for a better system that does apply methods for translation.

Chapter 3

Methods and Applications

3.1 Cryptology and Linear Transforms

Current approaches bring about an interesting school of thought in cryptology and cryptography. Encryption by code is generally defined as “A substitution crypto-system in which the plain-text elements are primarily words and sentences, and the code equivalents typically consist of letters and or digits in otherwise meaningless combinations of identical length”. This decryption system was done by both parties using Codebooks with look up reference tables of the words and their encrypted equivalents.[23] This systems describes a similar method being used and applied by Model 1 and leads to the idea that translation can be thought of as a form of encryption or decryption from one language to another.

Further still, encryption can be represented by linear mappings and in turn transformations, where the transformation takes plain-text to encrypted text by some transformation process. The comparison to translation can be made again by considering that translation is the transformation process and takes Source Language text to the Target Language. In general, this can be expressed as:

$$
\begin{gathered}
T : SL \to TL, \quad x \in SL, y \in TL \\
\bar{\boldsymbol{x}} = x_1 x_2 x_3 \dots x_k = (x_1, x_2, x_3, \dots, x_k) \\
\bar{\boldsymbol{y}} = y_1 y_2 y_3 \dots y_m = (y_1, y_2, y_3, \dots, y_m) \\
\bar{\boldsymbol{y}} = T(\bar{\boldsymbol{x}})
\end{gathered}
$$

Where $\bar{\boldsymbol{x}}$ is the source language text with k n-gram words and $\bar{\boldsymbol{y}}$ the target language equivalent with m n-gram words, allowing for the number of words to be different.

Using this representation for complex systems would be inefficient however as Model 1 is simple a linear mapping representation which would be given by,

$$
\begin{gathered}
T : SL \to TL, \quad x \in SL, y \in TL \\
\bar{\boldsymbol{x}} = x_1 x_2 x_3 \ldots x_k = (x_1, x_2, x_3, \ldots, x_k) \\
\bar{\boldsymbol{y}} = (y_1, y_2, y_3, \ldots, y_k) = T(\bar{\boldsymbol{x}}) = T(x_1, x_2, x_3, \ldots, x_k) \\
T : Dic_{SL}(x_k) = Dic_{TL}(y_m), \qquad Dic = \textit{Dictionary No.}
\end{gathered}
$$

For more complex systems the underlying transformation process would have various components representing the various subsystems that aim to function on different areas of the translation. Representing this with a single transformation would only work as a generalisation.

As we work under the idea that translation can be represented by linear transformations, imagining some generalised transformation that gives a correct translation for any given SL input. The aim of Machine translation can be redefined as finding methods to represent this transformation as efficiently as possible.

As Machine Translation works from the perspective that does not initially know the computation behind translation, the goal of MT can be further reduced to breaking the encryption/decryption of translation from one language to another. Methods for breaking encryption systems from cryptography can then be applied to machine translation with this thought process in mind. Not all applications will be relevant as MT does not work under the principle that either the SL or TL intends to be hidden nor is a language specifically built to be translated.

Chapter 4

Model 2

4.1 Statistical Translation

The dictionary translator model had limited success primarily rooted in the underlying logic rather than the model itself as it does little in actual translation, acting more as a substitution of words. Statistical Translators typically outperform exclusive rule-based translators as well as adding levels of complexity into the system. For a secondary model a statistical system was built, taking the logic applied from the first Model as well as the generalised concept outlined in the Types of Machine Translation section.

The aim behind SMT systems is to optimise for some most probable translation of a set of possible translations for some input sentence.

$$\hat{x} = \text{argmax}\{P(x|y)\} = \text{argmax}\,\{P(y|x)P(x)\}$$

$$\hat{y} = \text{argmax}\{P(y|x)\} = \text{argmax}\,\{P(x|y)P(y)\}$$

To construct a model, the definition for translation probability and set of possible translations must be defined.

4.1.1 Set of Translations

A set of possible translations would ideally only contain strings of text that could be given as translations for the SL input. To produce such a list would require translations of strings prior, therefore assumptions or additional subsystems must be applied.

Consider some set of translations where the translations given by Model 1 are elements. Such a set could be some combination of every possible translation of each SL input word. For each n-gram word in the SL sentence string, there is a finite number of word to word translations. Each word in the SL can be considered a vector containing all TL words it can be translated to. The vector $\bar{\boldsymbol{x}}$ of SL words would become a matrix where each row represents the list of translations for the n^{th} word of the SL string.

$$\bar{\boldsymbol{x}} = (\bar{\boldsymbol{x}}_1, \bar{\boldsymbol{x}}_2, \dots, \bar{\boldsymbol{x}}_k) = \begin{bmatrix} y_1^1 & y_1^2 \cdots y_1^N \\ y_2^1 & y_2^2 \cdots y_2^N \\ \vdots & \vdots \ddots \\ y_k^1 & y_k^2 \cdots y_1^N \end{bmatrix} \implies \left\{ \begin{matrix} \begin{bmatrix} y_1^1 & y_2^1 & \cdots & y_k^1 \end{bmatrix} \\ \begin{bmatrix} y_1^1 & y_2^1 & \cdots & y_k^2 \end{bmatrix} \\ \vdots \qquad\qquad \vdots \\ \begin{bmatrix} y_1^1 & y_2^1 & \cdots & y_k^N \end{bmatrix} \\ \vdots \qquad\qquad \vdots \\ \begin{bmatrix} y_1^1 & y_2^2 & \cdots & y_k^1 \end{bmatrix} \\ \vdots \qquad\qquad \vdots \\ \begin{bmatrix} y_1^N & y_2^N & \cdots & y_k^N \end{bmatrix} \end{matrix} \right\}$$

Where y_j^i is the i^{th} word in the SL string and the j^{th} possible translation for that word along with N denoting the last translation of any given x. The resultant set could be permuted further by rearranging word order as well but would cause a significantly larger sized set.

4.1.2 Translation Probability

The probability of a translation of some SL string, is the likelihood of the string being given as a translation out of the set of possible translations. Since the set of translations has been expanded to a of set combinations of translated words, the probability can be expanded as well. Optimally, the probability would be determined by the frequency of which the TL sentence string is given as a translation. As this would demand large data with finite differences, an alternative is to consider a comparison of frequency of translation with frequency of occurring in the TL to begin with.

Consider a string of text in a known language, broken down into all n-gram letters that occur in sequence starting with each letter working up to the entire string n-gram.

$$\bar{\boldsymbol{x}} = x_1, x_2, \dots, x_k \implies (x_1, x_2, \dots, x_k), (x_1x_2, x_2x_3, \dots, x_{k-1}x_k) \dots (x_1x_2 \dots x_k)$$

Each of these n-grams have an individual frequency of occurring in the language which can be applied as a measure that gives the frequency of any string of any length occurring in the language.

$$P(\bar{\boldsymbol{x}}) = P(x_1, x_2, ...x_k) = (f_1^1 \cdot f_2^1 ... f_k^1)(f_1^2 \cdot f_2^2 ... f_{k-1}^1) ... (f_1^k)$$

Where f_k^n is the frequency of the k^{th} n-gram. The product results in a measure that is weighted down for any combination of infrequent component n-grams in the string.

Although accuracy is given by using all possible n-grams, the possible permutations of n-grams as n increases grows by a factor of a the size of the alphabet. The total number of possible n-grams at any n is a^n. As n gets larger the average frequency decreases as a result of the expanded n-gram alphabet. In terms of the measure, higher n-gram frequencies contribute better accuracy by allowing for more levels of distinction, but a generalised probability can still be obtained by only considering lower n-gram components.

In machine translation and in terms of the set of translations for the TL, the n-grams used are in terms of words. A probability measure for the likeliness of occurring in the TL can apply this method and produce a product of the frequencies of n-gram words. The resulting translation would be a string that is likely in the TL based on the input SL but not necessarily a string that is equivalent. Similar applications to this have been applied in statistical translators focusing more on the deterministic probability aspect.[24]

4.2 Model Application

This can be applied in a model using a steepest accent algorithm, which are computational methods for finding the highest point in some sample set, such as the most probable in the set of generated sentence strings.

To keep the model simple Basic English has been used, but to produce better results the 1000-word variant has been chosen. In terms of assumptions, this model does not assume a one to one relationship but does assume that any translation can be represented by some combination of each individual word's possible translations. Although this is true in some cases, it ignores instances in which the SL and TL use different constructions or patterns to express something as well as ignoring idioms or any translation to omission.

4.2.1 Data

To produce this model data is required on frequency of n-grams for the TL Welsh, along with the possible translations for all words in the SL, Basic English. Data for n-gram frequency was obtained from CorCenCC[25] an open source corpus for contemporary Welsh and individual word frequency by an analysis done by Bangor University.[16] For the purposes of this model, only 1-gram and 2-gram frequencies of words are considered.

To obtain possible translations for Basic English a simple script was produced with the purpose of inputting and collecting all dictionary entries for each word.[26] A problem identified with the data is the indiscriminate inclusion of translations even if it is not word for word (for example, a translation for ***it*** would be ***yomorol***, literally **to leg *it***). Additionally, not all translations are possible leaving certain words without entries in the set of translations. Overall, this means the obtained set of data is imperfect and must be considered during its application.

A further issue identified involves the use of separate sources for frequency of Welsh words and possible Welsh translations. The dictionary used does not consider the grammatical case of Consonant Mutation seen in Celtic Languages where the first letter of a word will change dependent on grammatical rules. The frequency lists used do consider mutated words and count them separately to the root words. This can cause the issue of ranking a correct translation as improbable because the relevant mutation masks the frequency in the data.

4.3 Statistical Translation Model

A statistical translator for input sentences was created in python and the previously produced paragraph translator was modified to work with this model.

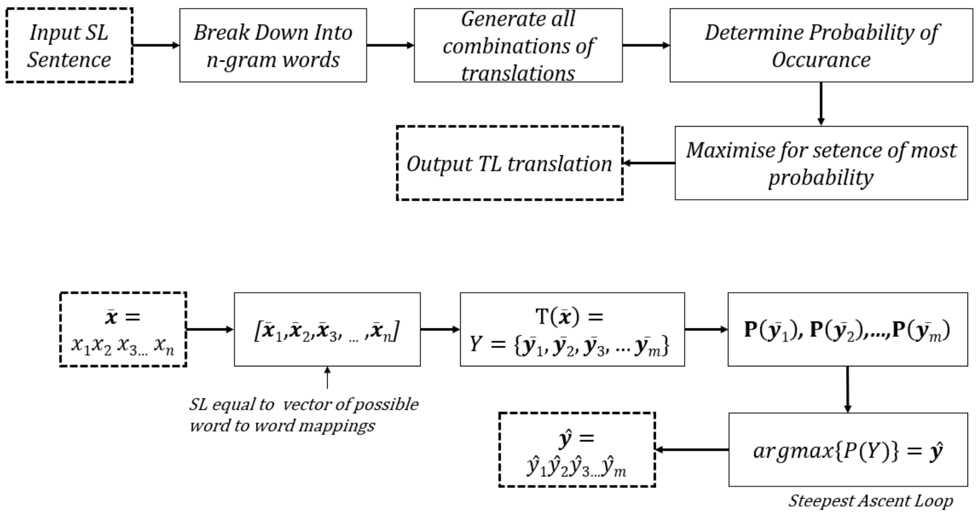

Input SL Sentence
Break Down Into n-gram words
Generate all combinations of translations
Determine Probability of Occurance
Maximise for setence of most probability
Output TL translation
$\bar{x} = x_1 x_2 x_3 \dots x_n$
$[\bar{x}_1, \bar{x}_2, \bar{x}_3, \dots, \bar{x}_n]$
SL equal to vector of possible word to word mappings
$T(\bar{x}) = Y = \{\bar{y}_1, \bar{y}_2, \bar{y}_3, \dots \bar{y}_m\}$
$P(\bar{y}_1), P(\bar{y}_2), \dots, P(\bar{y}_m)$
$argmax\{P(Y)\} = \hat{y}$
Steepest Ascent Loop
$\hat{y} = \hat{y}_1 \hat{y}_2 \hat{y}_3 \dots \hat{y}_m$

Figure 4.1: Statistical Translator

The Statistical Translation function is split into three function parts, the main function taking the SL input, a function to generate the combinations of translated sentence strings and a function that calculates a probability for a Welsh sentence string. The probability calculated uses the frequency of occurrence for up to 2-gram words and to limit the sentence generation from being too large, the combinations do not permute word order.

4.4 Results

$$\boxed{\textit{go to the shop} \xrightarrow{\textbf{English}\rightarrow\textbf{Welsh}} \textit{mynd i 'r siop}}$$

Figure 4.2: Statistical Translator Sentence Example

$$\boxed{\begin{array}{l} \textit{the cat is on the floor.} \\ \textit{the dog is not here} \end{array} \xrightarrow{\textbf{English}\rightarrow\textbf{Welsh}} \begin{array}{l} \textit{y sydd cath ar y llawr.} \\ \textit{y ci sydd ni yma.} \end{array}}$$

Figure 4.3: Statistical Translator Paragraph Example

Unlike the first Model, compatibility for words that do not occur in the SL is not built in and therefore only words in basic English can be used. A system for printing the number of sentences generated exists giving, 3150 , 11088, 48 generated sentences for *Figure* 4.2 and *Figure* 4.3 respectively.

4.5 Review

In terms of the Model, simple inputs can yield correct grammatical translations and complex inputs can yield partially correct grammatical translations. The time taken to translate scales with the number of sentences generated and can take some time.

In terms of translation overall, the translations can produce some accurate grammar. The model does not identify contractions causing the problem seen in *Figure* 4.2, where a space exists between two words that should be contracted, ***i'r***. There are also problems related to translation to omission and translation to lacking dictionary entries causing no translation to be given for the entire sentence.

4.6 Improvements

The largest limiting factor is the quality of data used as the overall application works well enough to produce a level of translation. Improvements to the set of possible words could fix the problems with lacking dictionary entries, translations to multiple words as well as translation to omitted words. Higher quality would also been seen with using data for higher n-gram frequency and flexibility would be seen by allowing words that are not in Basic English to be carried through without translation.

Chapter 5

Measures for Translation Quality

To gauge the quality of translations given by a machine translator a form of quality testing is required. Quality measures exist for this purpose that work by weighing different aspects of the translations given by the systems against reference translations that have been produced by hand. The idea behind these measures is that they should correlate with the opinions given by people and speakers of the languages such that a good score would imply the same from a fluent speaker.

Measures can focus on different areas of the translation, Bilingual Evaluation Understudy (BLEU) focuses on words being correctly translated to a correct amount whereas the Word Error Rate (WER) focuses on the distance between the reference and translation.[27]

Quality measures can be calculated on different axes to judge different aspects of the translations. As translation is a one to many relationship, using multiple references or focusing on the closest reference to the translation assures the translation is judged fairly. It can also be broken down into n-gram words and comparing that with the same of the references, judging the correctness of word order as well as the flow of the sentences.

5.1 Word Error Rate

The Word Error Rate, originally developed for measuring the quality of speech recognition, can be generalised as the distance or steps needed to take the translation to the reference out of the number of words in the reference. The possible steps to take a word from the translation to the reference include, substitution S, deletion D or insertion I. Where the calculation is given by: [29]

$$WER = \frac{I + S + D}{m}$$

where m is the number of words in the reference sentence. This in turn means the measure can be larger than 1 and that a better score is a lower score.

The calculation of insertions, deletions and substitutions comes from the Levenshtein distance from information theory which is a metric for measuring the distances between strings. By first considering n-gram letters the Levenshtein distance can be calculated by constructing a matrix.

For two words of size M and N, construct an $M+1$ by $N+1$ matrix with the first column and row containing 0 to M and 0 to N respectively.

$$L = \begin{bmatrix} 0 & 1 & 2 & \dots & N \\ 1 & l_{22} & l_{23} & \dots & l_{2N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ M & l_{M2} & l_{M3} & \dots & l_{MN} \end{bmatrix} = \begin{array}{c|ccccc|} & & x_1 & x_2 & \dots & x_N \\ & 0 & 1 & 2 & \dots & N \\ y_1 & 1 & l_{22} & l_{23} & \dots & l_{2N} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ y_M & M & l_{M2} & l_{M3} & \dots & l_{MN} \end{array}$$

Where $y_1 ... y_M$ and $x_1 ... x_N$ represents the letters of the words. Each remaining element beginning with row 2 column 2 can be calculated by the following algorithm:

```
for i in range 2 to M
    for j in range 2 to N
        if Column letter equals Row letter:
            Sub = 0
        else:
            Sub = 1
        $L_{ij} = \min\{L_{[i-1][j]} + 1, L_{[i][j-1]} + 1, L_{[i-1][j-1]} + Sub\}$
```

The distance is then given by the element l_{MN} in bottom right which is the minimum edit distance or in terms of the original WER is equal to the sum of substitutions, insertions and deletions.

This can be applied to n-gram words without the need to modify the algorithm and would allow the equation for word error to be modified to:

$$WER = \frac{l_{km}}{m}, \ l \in L$$

Where k is the number of words in the translation sentence and m the number in the reference sentence.

5.2 BLEU

The BLEU method, fashioned from the Word Error Rate, is best described by considering an example sentence and a reference:

Candidate:	*The cat is on the floor*
Reference:	*The cat sat on the floor*

The measure takes a sum of the number of words from the candidate that appear in the reference out of the number of words in the reference. In the case of the above example

$$\text{the} \times 2, \text{cat} \times 1, \text{sat} \times 0, \text{on} \times 1, \text{floor} \times 1 \implies \frac{2+1+0+1+1}{6} = \frac{5}{6}$$

To avoid the problem where a word appearing in the candidate more than the reference being rewarded simply because it is in the reference, each word is limited to contributing the number of times it appears in the reference to the sum. This could be considered as taking the minimum of the number of times a word appears in the reference with the number of times it appears in the candidate.

Consider a sentence vectors $\bar{\boldsymbol{x}}$ and $\hat{\boldsymbol{x}}$ representing the candidate and reference,

$$\bar{\boldsymbol{x}} = x_1 x_2 x_3 \dots x_k = (x_1, x_2, x_3, \dots x_k)$$
$$\hat{\boldsymbol{x}} = \hat{x}_1 \hat{x}_2 \hat{x}_3 \dots \hat{x}_m = (\hat{x}_1, \hat{x}_2, \hat{x}_3, \dots \hat{x}_m)$$

Let $\hat{X}$ represent the set of unique words of the reference vector such that

$$\hat{X} = \{\hat{x}_i | \forall \hat{x}_i \in \hat{\boldsymbol{x}}\}, \quad |\hat{\boldsymbol{X}}| = M$$

Let the number of occurrences of some n-gram word $\hat{x}_i$ from the set X in the vectors $\bar{\boldsymbol{x}}$ and $\hat{\boldsymbol{x}}$ be c_i and $\hat{c}_i$ respectively. The BLEU measure of translation can then be given as,

$$BLEU = \frac{1}{k} \sum_{i=1}^{M} \min\{c_i, \hat{c}_i\}$$

This measure gives a number between 0 and 1 where a better score is a higher score unlike the Word error rate system.

5.3 Quality Results

Model 1 The Dictionary translator and Model 2 The Statistical translator were tested for quality and compared using Word Error Rate and BLEU. Twenty-one reference translations were obtained from parallel bilingual texts that would fit the constraint language of Basic English. These sentences were sourced from articles from the Bilingual Welsh magazine website Parallel Cymru as well as small sections from proceedings of the National Assembly for Wales. Sentences were additionally produced by hand by translating sentences from Seuss' The Cat in the Hat.[31][32][22]
As only singular reference sentences could be obtained per translation the measures have been produced using varied n-grams to allow for a better range of comparisons. BLEU has been measured for n-grams 1 to 4 and WER for 1 to 2. The former has been chosen as $n = 4$ has been shown to correlate highly with language for BLEU. The average scores per n-gram can be seen in *Figure* 5.1 and *Figure* 5.2 below and a full list of the sentences can be found in the appendix, section 8.1.

	$n = 1$	$n = 2$	n-gram Avg
Model 1:	72.38%	96.19%	84.29%
Model 2:	65%	93.75%	79.375%

Figure 5.1: Word Error Rates Model 1 and 2 for $n = 1, 2$

	$n = 1$	$n = 2$	$n = 3$	$n = 4$	n-gram Avg
Model 1:	44.37%	9.84%	3.43%	0%	14.41%
Model 2:	56.7%	9.06%	0%	0%	16.44%

Figure 5.2: Word Error Rates Model 1 and 2 for $n = 1, 2, 3, 4$

These results show there is a measurable improvement from Model 1 to Model 2 with the overall average WER trending down and the overall average BLEU trending up. It should be noted, due to lacking entries in the translations for Model 2 some translations given are blanks and as this is a problem with the data and not the system these were excluded from the results during calculation.

Chapter 6

Model 3

6.1 Hybrid Translation

Unlike Model 1, the second Model's success was rooted in a limited form of translation but was largely restricted by the data as well as giving an output that can still be improved. A logical next step would be to produce a Hybrid translator, using the mechanics behind Model 2 with improved data and applying rule-based proofing to the generated output.

6.1.1 Rules

The possible rules that can be implemented could be split into two groups of focus, input and output. Where the rules for input contribute with the translation or towards optimising the system itself such as by decreasing the number of sentences needing to be checked. Alternatively, rules for output would focus on proofing by algorithmically checking and correcting common grammatical problems or problems that are caused by the Machine Translator itself.

6.2 Data

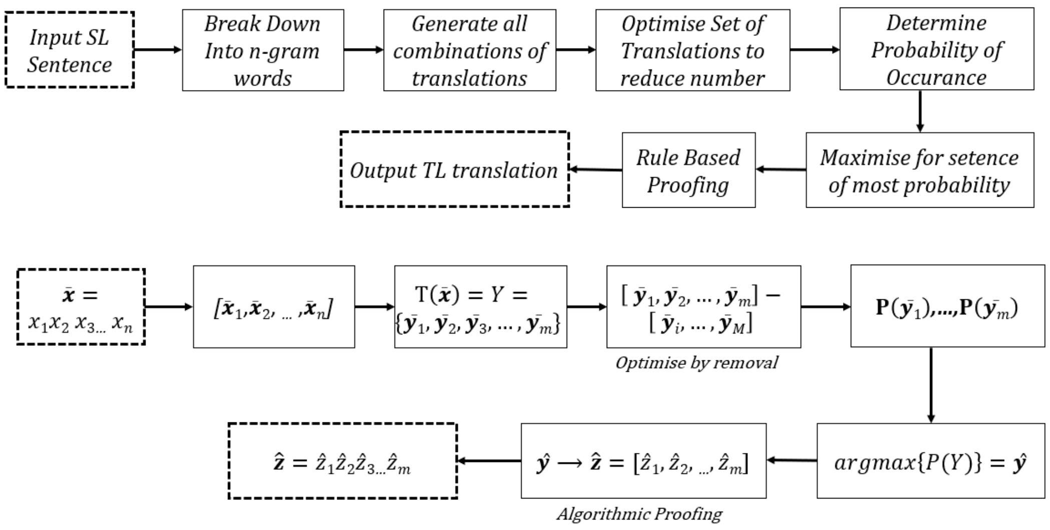
To obtain higher quality data similar scripts were used with two Welsh and English dictionaries[26][33] but with added filters to ensure relevant translations as well as translations to multiple words being stored with underscores to be properly broken down later. The two sets of translations obtained were combined into one to optimise where one dictionary may lack an entry the other gave. The result was a more robust and populated set but still with some missing entries where neither dictionary could give a translation.

Data for n-grams were sourced again from CorCenCC[25] using $n = 2, 3$ to improve the calculated probability as well as the individual word frequency analysis done by Bangor University.[16]

One of the most expanded forms of Basic English is a 1800-word variant that contains all of the words from the previously used forms of Basic English. Using this as the constraint language allows for more flexible SL input and again brings us closer to English whilst keeping SL simple.

6.3 Translation System

A hybrid translator was produced in python based on the system of model 2. The previously produced paragraph translator was modified to work with this system as well.



Input SL Sentence
Break Down Into n-gram words
Generate all combinations of translations
Optimise Set of Translations to reduce number
Determine Probability of Occurance
Maximise for setence of most probability
Rule Based Proofing
Output TL translation
$\bar{x} = x_1 x_2 x_3 \dots x_n$
$[\bar{x}_1, \bar{x}_2, \dots, \bar{x}_n]$
$T(\bar{x}) = Y = \{\bar{y}_1, \bar{y}_2, \bar{y}_3, \dots, \bar{y}_m\}$
$[\bar{y}_1, \bar{y}_2, \dots, \bar{y}_m] - [\bar{y}_i, \dots, \bar{y}_M]$
Optimise by removal
$P(\bar{y}_1), \dots, P(\bar{y}_m)$
$argmax\{P(Y)\} = \hat{y}$
$\hat{y} \longrightarrow \hat{z} = [\hat{z}_1, \hat{z}_2, \dots, \hat{z}_m]$
Algorithmic Proofing
$\hat{z} = \hat{z}_1 \hat{z}_2 \hat{z}_3 \dots \hat{z}_m$

Figure 6.1: Hybrid Translator

The core systems from Model 2 have been kept with the modification to 3-gram word frequency included in the probability equation. The Sentence generation function calls an optimisation function that looks for two common grammatical cases in Welsh which change form dependent on simple grammatical rules. By converting all these cases to one form, all duplicate sentences can be removed, and the correct forms can be implemented after probability calculation.

$$\textbf{the} = \{y,\ yr,\ \text{‘}r\} \to \{y\}$$
$$\textbf{in} = \{yn,\ ym,\ yng\} \to \{yn\}$$

The case chosen to swap to is the most frequent and as this is done for all generated sentences there is no effect on the outcome of the most probable sentence. Further optimisation is done by the deletion of any sentence starting with an apostrophe. These are the result of contractions which do not occur if there is no proceeding word before the apostrophe in the sentence.

A rule-based proofer is called after a most probable sentence has been produced. This algorithmically checks for the grammatical cases which can be optimised in order to fix after probability calculation. A simple consonant mutation system has been implemented also. Welsh as well as other Celtic languages have special grammatical rules known as mutations where after certain words one of three mutations will happen causing the initial consonant letter of the successive word to change in an algorithmic fashion. The implemented system is not exhaustive but checks for common mutations

6.4 Results

go to the shop $\xrightarrow{\textbf{English}\rightarrow\textbf{Welsh}}$ *mynd i'r siop*

Figure 6.2: Hybrid Translator Sentence Example

The sun did not shine. it was too wet to play. so we sat in the house. all that cold cold wet day $\xrightarrow{\textbf{English}\rightarrow\textbf{Welsh}}$ *Yr haul **did** ni disgleirio. hi was hefyd gwlyb yn canu. fel ni **sat** yn y ty. pob a oer oer gwlyb dydd*

Figure 6.3: Hybrid Translator Paragraph Example

The highlighted errors in *Figure* 6.3 are a result of tense not being considered as dictionaries typically only translate present tense forms of words between English and Welsh. This is due in part to tense being denoted by different constructions in Welsh and English.

6.5 Review

In terms of the assumptions of the Model, simple and complex inputs can yield entirely correct grammatical translations, but complex inputs are still more likely to be partially incorrect. The optimisation rules reduced the computational effort necessary generally ensuring a better time frame for each translation.

In terms of translation as a whole, even though this is still a simple Model the end result correlates high enough with a translation because of the rule-based proofing as well as the improved data that this model appears to exhibit an actual level of translation.
The system is still limited by tense, word order and lacking translation entries but overall performs as a simple translator.

6.6 Improvements

Further improvements to this system could be as simple as applying more rules for proofing and optimisation, or more complicated focusing on aspects of the data or underlying systems. Seeking data on forms of words in different tenses could help with the problem of non-present tense forms of words not being translated.
Tense is often denoted by different constructions in languages and is true for Welsh and English meaning it might be better fixed by exploring a system that checks for tense in the SL input in order to correct it in the output algorithmically. This would require data on constructions and methods of denoting tense in both languages.

Another improvement would be to the choice of only finding translations of the constraint language words individually as it is possible that combinations of words in English have a specific translation such as with,

in a = *mewn*

to run = *rhedeg*

Collecting data on possible combinations of words and obtaining their translation can be worked into this system organically but would require a significantly larger pool of words to be translated by dictionary first.

6.7 Translation Quality

Using the previously applied BLEU and Word Error Rate the quality of this model can be determined as well as a comparison to Model 1 and 2 to see how effective the modifications have been. This can be seen in *Figure* 6.4 and *Figure* 6.5 as well as a comparison for the same set of references and measures with Google Translate.

	$n = 1$	$n = 2$	n-gram Avg
Model 1:	72.38%	96.19%	84.29%
Model 2:	65%	93.75%	79.375%
Model 3:	56.92%	79.75%	68.33%
Google:	30.33%	43.92%	37.13%

Figure 6.4: Word Error Rates Model 1,2,3 and Google Translate for $n = 1, 2$

	$n = 1$	$n = 2$	$n = 3$	$n = 4$	n-gram Avg
Model 1:	44.37%	9.84%	3.43%	0%	14.41%
Model 2:	56.7%	9.06%	0%	0%	16.44%
Model 3:	55.73%	25.95%	18.45%	6.67%	26.7%
Google:	16.08%	62.38%	49.52%	29.17%	39.29%

Figure 6.5: Word Error Rates Model 1,2,3 and Google Translate for $n = 1, 2, 3, 4$

A clear trend can be seen of WER decreasing and BLEU increasing overall which shows a trend of improvement throughout each model. The 4-gram BLEU gives a non-zero measure for Model 3 which has been shown to be the most representative of language suggesting that this model has a measurable level of actual translation.

Comparing all three models with Google Translate using the same reference sentences, it is clear the models are far simpler and that there is far more that can be done to improve the systems.

Chapter 7

Conclusion

7.1 Further Methods and Alternative Applications

Various ideas and applications were considered in this project many of which proved inefficient or too complex. One approach saw an attempt to find matrix representations of the linear transformations in a way that could be manipulated using logic from linear algebra such as considering a vector space. Other applications investigated included co-occurrence matrices, graph-based applications as well as higher end statistical methods focusing on distributions rather than only n-gram frequencies.

There is also opportunity to further explore Lexical similarities to obtain a more refined measure. It was originally attempted to apply the RMSE in terms of words but attempts to rank the data by shared entries proved a problem due to the one to many relationships of translation. Additionally, a similar approach could be made to tackle phonetic lexical similarity by first reducing the words into phonetic components using the international phonetic alphabet.

7.1.1 Neural Network Translation

A logical next step for a fourth model would be to consider the remaining type of machine translation Neural Networks. The training of a Neural Network MT would require access to large corpora of parallel texts in Welsh and English the data for which is limited.

An alternative approach considered was to train such a translator using the existing Models in order to understand this area of MT by using the areas that have already been explored. Additionally, the previously applied translation quality methods could be used in this system to contribute a way for the Neural Network to judge success during each iteration.

7.2 Project aims

This project aimed to understand fundamental areas of machine translation in order to apply them in the construction of simple machine translators. Areas such as types of MT, lexical similarity, probability and measures for translation have been explored. Comprehensive logic has been used to approach and explain the steps to these ideas as well as to build on them in ways to simplify their application.

The resultant translators produced led to a final Model that measured a limited level of actual translation suggesting the approaches taken have led to a successful understanding of core elements of the field. Steps have been taken in order to represent the understanding using mathematical logic and notations which in turn have been used to explain all applications and models. To fully represent the translational capacity and necessary input restrictions of the final model an example text paragraph has been translated with the SL text, modified SL and TL translation in the Appendix, section 8.2.

The underlying aim to outline areas of difficulty for building translators for a minority language can be discussed for that of Welsh. Information, resources and data for probabilities and frequencies were easy to come by for English but hard for Welsh. The predominant improvement through each Model was to improve the data used which is difficult for Welsh as only limited data exists. The data sources used were largely imperfect and collected for different reasons with different assumptions causing them to fit together in ways that left areas of problem such as tense and word order.

To efficiently build machine translators for a minority language access to larger and higher quality sets of data is necessary in order to produce a system that can accurately mimic the properties of the language.

Bibliography

[1] Machine Translation - Warren Weaver Memorandum - Rockefeller Foundation
http://www.mt-archive.info/Weaver-1949.pdf
Cambridge, Massachusetts: MIT Press. (1949)

[2] Language and machines computers in translation and linguistics - Automatic Language Processing Advisory Committee
http://www.mt-archive.info/ALPAC-1966.pdf National Academy of Sciences, Washington, D. C. 1966

[3] ALPAC The in(famous) report - John Hutchins
http://www.hutchinsweb.me.uk/ALPAC-1996.pdf
Cambridge, Mass: The MIT Press, 2003

[4] The history of machine translation in a nutshell - John Hutchins
http://hutchinsweb.me.uk/Nutshell-2005.pdf
Latest revision: November - 2005

[5] Ten years of Google Translate - Barak Turovsky
https://blog.google/products/translate/ten-years-of-google-translate/
Google Blog - 2016

[6] Theoretical Overview of Machine translation - Mohamed Amiee Chéragui
http://ceur-ws.org/Vol-867/Paper17.pdf
African University, Adrar, Algeria - 2012

[7] The Mathematics of Statistical Machine, Translation: Parameter Estimation
- Peter E Brown,Stephen A. Della Pietra, Vincent J. Della Pietra, Robert L. Mercer
https://www.aclweb.org/anthology/J93-2003.pdf
African University, Adrar, Algeria - 2012

[8] Machine translation: a concise history - John Hutchins
http://www.hutchinsweb.me.uk/CUHK-2006.pdf
African University, Adrar, Algeria - 2012

[9] Neural Machine Translation by Jointly Learning to Align and Translate
- Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio
https://arxiv.org/abs/1409.0473
Universite de Montreal - 2016

[10] English and Welsh - J. R. R. Tolkien
https://faculty.smu.edu/bwheeler/tolkien/online_reader/T-English&Welsh.PDF
Inaugural O'Donnell Memorial Lecture - October 21, 1955

[11] Systems in Welsh Phonology - Alan R. Thomas
https://search.proquest.com/openview/5c13d384d9a4e0d57dce43a8b1cac943/1?cbl=1819581&pq-origsite=gscholar
University of Bangor - 1966

[12] Cross-linguistic Similarity in Foreign Language Learning - Håkan Ringbom
https://www.researchgate.net/publication/326866841_Cross-linguistic_Similarity_in_Foreign_Language_Learning
2006

[13] Business Dynamics, System Thinking and Modeling for a Complex World (errata) - John Sterman
https://www.researchgate.net/publication/44827001_Business_Dynamics_System_Thinking_and_Modeling_for_a_Complex_World
Massachusetts Institute of Technology - 2000

[14] What does RMSE really mean? - James Moody (Blog Post)
https://towardsdatascience.com/what-does-rmse-really-mean-806b65f2e48e
Towards Data Science - 2019

[15] English Letter Frequency Counts: Mayzner Revisted
https://norvig.com/mayzner.html

[16] Cronfa Electroneg o Gymraeg (CEG): A 1 million word lexical database and frequency count for Welsh - Ellis, N.C., O'Dochartaigh, C, Hicks, W, Morgan, M., & Laporte, N.
www.bangor.ac.uk/canolfanbedwyr/ceg.php.en
Bangor University - 2001

[17] Frequency of German Words Data
https://github.com/gambolputty/dewiki-wordrank

[18] Frequency of French Words Data
https://wortschatz.uni-leipzig.de/en/download

[19] The first public demonstration of machine translation: the Georgetown-IBM system - John Hutchins
http://www.hutchinsweb.me.uk/GU-IBM-2005.pdf
Expanded version of AMTA-2004 paper. Written November 2005

[20] Basic English - Charles K Ogden
http://ogden.basic-english.org/be0.html
1925

[21] Ronald E. Wyllys - Empirical and Theoretical Bases of Zipf's Law
https://www.ideals.illinois.edu/bitstream/handle/2142/7182/librarytrendsv30i1g_opt.pdf
University of Texas at Austin - 1981

[22] Cat in the hat poem - Dr Seuss
https://www.oatridge.co.uk/poems/d/dr-seuss-cat-in-the-hat.php

[23] A History of U.S. Communications Security (Volumes I and II) - the David G. Boak Lectures, NSA
https://www.governmentattic.org/18docs/Hist_US_COMSEC_Boak_NSA_1973u.pdf
The National Security Agency - 1973

[24] N-gram-based Machine Translation- Jose B.Marino, Rafael E. Banchs, Josep M. Crego, Adria de Gispert, Patrik Lambert, Jose A. R. Fonollosa, Marta R. Costa-jussa
https://www.mitpressjournals.org/doi/pdf/10.1162/coli.2006.32.4.527
Universitat Politecnica de Catalunya - 2006

[25] CorCenCC Corpus - Open Source Corpus for Contemporary Welsh Language
https://corpusdemo.corcencc.org/home?language=en

[26] Online Welsh to English dictionary
https://geiriadur.uwtsd.ac.uk/
Trinity Saint David University

[27] Minimum Error Rate Training in Statistical Machine Translation - Franz Josef Och
https://www.aclweb.org/anthology/P03-1021.pdf
University of Southern California - 2003

[28] BLEU: a Method for Automatic Evaluation of Machine Translation -
Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu
https://www.aclweb.org/anthology/P02-1040.pdf
IBM - 2002

[29] On the Use of Information Retrieval Measures for Speech Recognition Evaluation -
Iain McCowan, Darren Moore, John Dines, Daniel Gatica-Perez,
Mike Flynn, Pierre Wellner, Herv'e Bourlard
*https://www.researchgate.net/publication/37433359_On_the_Use_of_Information_Retrieval_Measures
_for_Speech_Recognition_Evaluation*
IDIAP Research - 2005

[30] A Linear Space Algorithm for Computing Maximal Common Subsequences
- D.S Hirschberg
https://www.ics.uci.edu/ dan/pubs/p341-hirschberg.pdf
Princeton University - 1975

[31] Parallel Cymru - Bilingual Welsh Online
https://parallel.cymru/

[32] Phrase-based statistical - machine translation between English and Welsh - Jones, D.
http://xixona.dlsi.ua.es/corpora/UAGT-PNAW/
Universitat d'Alacant, Grup Transducens - 2006

[33] Geriadur Bangor (English Welsh Dictionary)
http://geiriadur.bangor.ac.uk/
Bangor University

[34] LEARNING BASIC ENGLISH: A Practical Handbook for English-Speaking People -
I. A. Richardsons and Christine Gibson
https://archive.org/details/in.ernet.dli.2015.166025

Chapter 8

Appendix

8.1 Translation Quality Sentences

Welsh Reference	**English Reference**
Gwelwch chi yna	See you there
Mae ci ar y lawr	There is a dog on the floor
Mae fy ngath yn glas	My cat is blue
Dw i'n hoffi coffi	I like coffee
Dw i ddim yn hoffi te	I do not like tea
Y gath du a'r ci du	The black cat and the black dog
Haul coch a lleuad gwen	Red sun and white moon
y gath yn yr het	the cat in the hat
am y dydd	for the day
nawr nawr cael dim ofn	now now have no fear
gyda chwpan a cacen	with a cup and a cake
y llaeth a'r cacen	the milk and the cake
o lan yna ar y pel	from up there on the ball
nawr edrychwch ar y ty 'ma	now look at this house
miliwn eleni a	million this year and
miliwn yn i gyfanswm o	million in to a total of
fy ngwraig	my wife
croeso yma	welcome here
nos da	good night
bore da	good morning
dw i ddim yn bwyta hynny	i do not eat that

8.2 Translation Example

In order to represent the translation capacity of the final model an example of text in Basic English taken from "LEARNING BASIC ENGLISH: A Practical Handbook for English-Speaking People" has been translated.[34]

Source Language Passage:
"When I had had my grain, I took a look round. In the box nearest me was a little fat gray horse, with thick hair on his neck and tail, a beautiful head and a questioning little nose. I put my head up to the iron rails at the top of my box, and said, "How do you do? What is your name?" He put his head round as far as his headband would let him, lifting it high, and said. "My name is Merrylegs. I am very good-looking. I take the young women on my back, and sometimes I take Mrs. Gordon out in the little low carriage. They have a high opinion of me, and so has James. Are you going to be in the box near me?" I said, "Yes.' "Then," said he, "it is my hope that you are good-humored; I would not be pleased to have anyone near me who has a trick of biting." "

Modified Source Language Passage:
"When I had had my grain. I took a look round. In the box nearest me was a little fat gray horse. with thick hair on his neck and tail. a beautiful head and a questioning little nose. *I put my head up to. the iron rails at the top of my box.* and said. How do you do. What is your name. He put his head round. as far as his headband would let him. lifting it high and said. My name is Merrylegs. I am very good looking. I take the young women on my back. and sometimes I take Mrs. Gordon out in the little low carriage. They have a high opinion of me. and so has James. Are you going to be in the box near me. said Yes. Then said he it is my hope that you are good humoured.*I would not be pleased to have anyone near me. who has a trick of biting.*"

Translation:
"Pan mi had had fy gronyn. mi took a edrych cylch. yn y bocs nearest i was a bach bras gray ceffyl. gan drwchus gwallt ar ei gwddf a cwt. a teg pen ac a questioning bach trwyn. mi rhoi fy pen lan yn. yr haearn rails yn y pen o fy bocs. a said. sut gwneud chi gwneud. beth yw eich enw. o rhoi ei pen cylch. fel pell fel ei headband would gadael o. lifting hi mawr a said. fy enw yw merrylegs. mi am iawn da looking. mi dal yr ifanc women ar fy cefn. a weithiau mi dal mrs. gordon allan yn y bach isel cerbyd. they cael a mawr barn o i. a fel has james. are chi going yn bod yn y bocs wrth i. said oes. hynny said o hi yw fy gobaith a chi are da humoured. mi would ni bod bodlon yn cael unrhywun wrth i. yr has a sgil o biting."

Note: The input text was modified by hand to replace punctuation with period marks as well as split two sentences up (italicised) to ensure the paragraph could be translated at a reasonably calculable level. This translation processed over a 1 hour time period.

8.3 Python Code

```
# Created: 10/12/2019
# Made by Rhys Nicholas
# Sentence Translator
# from Mathematical Approaches to Machine Translation
#
# Requires text files: cy.txt, en.txt

def DTrans(Sentence): #Define function: Dtrans [Dictionary Translation]
    Words = Sentence.split()  #Convert Sentence to a list of words
    Trans = [] #Define List for translations to be added to

    #For each word collect corresponding words
    for i in range(len(Words)):
        with open("en.txt","r") as en:  #Open txt English
            enD = en.read().split("\n") #Create list of words based on each line of English txt
            with open("cy.txt","r", encoding='utf-8-sig') as cy: #Open Welsh
                    cyD = cy.read().split("\n") #Create list of words based on each line of
            if Words[i].casefold() in enD: #If current Word in Loop is in English txt
                #Append corresponding Welsh word
                Trans.append(cyD[enD.index(Words[i].casefold())])
            else: #If Current word not found
                Trans.append(Words[i])

    return(' '.join(Trans)) #Print Translation with blank space for each word
```

```
# Created: 7/03/2020
# Made by Rhys Nicholas
# Statistical Sentence Translator
# from Mathematical Approaches to Machine Translation
#
# Requires text files: en.txt, 1Gram.txt, 2Gram.txt,
#                      P-1Gram.txt, P-2Gram.txt, Tmatrix.txt

#Useful for combinations and Permutations
import itertools

#Model 2 Statistical translator
#Main function
def STrans(Sen):
    Words = Sen.split() #Convert input to Vector
    Trans = [] # Vector for translation
    Matrix = [] #Full matrix of all translation words
    Tmatrix = [] #Matrix for Sentence Generation

    #Open Translation Matrix and read to placeholder variable
    with open("Tmatrix.txt","r") as f:
        T = f.read().split("\n")

    #Convert Translation Matrix from Place holder to Matrix (List of Lists)
    for i in range(len(T)):
        Matrix.append(T[i])
       # print(Matrix)

    #Import all Source Language (SL) words: 1000 Basic English
    with open("en.txt","r") as en:
        enD = en.read().split("\n")

    #Create New matrix in terms of the input SL
    #Each nth row represents the possible translations of the nth word
    for i in range(len(Words)):
        Tmatrix.append(Matrix[enD.index(Words[i])].split())

    #Create permutations of possible translations of words
    #Uses SenGen function
    SMatrix = SenGen(Tmatrix)
    print("Generating:", len(SMatrix), "Possible Sentences")#Print number of generations

    #Steepest Acsent Algorithm
    P = 0  #Start with probability 0 (minimum)
    for i in range(len(SMatrix)):   #For each Sentence
        if Prob(SMatrix[i])>P:      #Determine Probability
            P = Prob(SMatrix[i])    #Retain probability if higher than previous
            Trans = SMatrix[i]

    return " ".join(Trans)  #Return resultant string

#Model 2: Sub Function
#Generates combinations of sentences
def SenGen(Matrix): #Takes input List of Lists (Matrix)
    #Create permutations and store to placeholder variable
    Gen = list(itertools.product(*Matrix))
    SMatrix = [] #Permutation Matrix variable

    #Called function for permutations uses tuples
    #Loop through elements, convert to list, store in Permutation Matrix variable
    for i in range(len(Gen)):
        SMatrix.append(list(Gen[i]))

    return SMatrix #Return Permutation Matrix
    #Note does not produce arrangments therefore not true permutation



#Model 2: Sub Function
#Calculate Probability of a vector sentence
def Prob(Vec): #Takes input list of words, representing a sentence string

    #Data is finite therefore if no data is available
    #Assume probability is less than data obtained and use Fail Data
    TFail = 0.0001 #Fail Probability of 1-gram (words)
    OFail = 0.00000009 #Fail Probability of 2-gram (pairs of words)

    ##Following Data is organised such that nth elements
    ##will be corresponding per ngram
    ##E.g: The nth element of the 2gram probability list is
    ##the probability of the nth 2gram element word
    #Open 2 Gram Data store to variable
    with open("2Gram.txt","r") as f:
        TwoGram = f.read().split("\n")
    #Open 2 Gram probability Data store to variable
    with open("P-2Gram.txt","r") as f:
        PTwo = f.read().split()
    #Open 1 Gram Data store to variable
    with open("1Gram.txt","r") as f:
        OneGram = f.read().split()
    #Open 1 Gram probability Data store to variable
    with open("P-1Gram.txt","r") as f:
        POne = f.read().split()

    ##Decomposes input sentence into 2-gram words:
    ##E.g: go to shop => (go to, to shop)
    TVec=[] #Vector/List for Storing 2 Grams from sentence input
    #Create 2 grams of string input
    for i in range(len(Vec)-1):
        #There are N-1 possible 2-grams for N words

        #Join the elements together with " " space, to match the Data
        TVec.append(" ".join([Vec[i],Vec[i+1]]))


    ##Calculate 2Gram Probability
    PT = [] #Store probability of each 2gram
    for i in range(len(TVec)):  #For each 2gram
        if TVec[i] in TwoGram:  #If probability exists store it
            PT.append(float(PTwo[TwoGram.index(TVec[i])]))
            #Expanlation: Append the probability value of current
            #             word from probability data to Storage
            #             using the index of the current 2-gram word
            #             from position in the 2-gram data

        else:
            PT.append(TFail)    #Else use Failure Probability
```

```
##Calculate 2Gram Probability
PO = [] #Store probability of each 1gram
for i in range(len(Vec)):   #For each 1gram
    if Vec[i] in OneGram:   #If probability exists store it
        PO.append(float(POne[OneGram.index(Vec[i])]))
        ##Explanation: Refer to 2gram

   #Else use Failure Probability
    else:
        PO.append(OFail)

P = 1 #Let Probability = 1
for i in range(len(PO)): #Multiply by all 1gram probability
    P = P*PO[i]
for j in range(len(PT)): #Multiply by all 2gram probability
    P = P*PT[j]

return P #Return Resulting Probability
```

```
# Created: 16/03/2020
# Made by Rhys Nicholas
# Statistical Sentence Translator
# from Mathematical Approaches to Machine Translation
#
# Requires text files: en.txt, 1Gram.txt, 2Gram.txt, 3Gram.txt
#                      P-1Gram.txt, P-2Gram.txt,P-3Gram.txt TMatrix.txt


#Useful for combinations and Permutations
import itertools

#============Statistical Translation Functions============#
#Model 3 Hybrid translator
#Main function
def HTrans(Sen):

    Trans = [] # Vector for translation
    Matrix = [] #Full matrix of all translation words
    Tmatrix = [] #Matrix for Sentence Generation

    #Allows capital first letter to carry through
    Caps = 0 #Capital First letter
    if ord(Sen[0]) < 91 and ord(Sen[0]) >64:
        Caps = 1
    Sen = Sen.lower() #Strip any uppercase
    Words = Sen.split() #Convert input to Vector

    #Open Translation Matrix and read to placeholder variable
    with open("TMatrix.txt","r") as f:
        T = f.read().split("\n")

    #Convert Translation Matrix from Place holder to Matrix (List of Lists)
    for i in range(len(T)):
        Matrix.append(T[i])

    #Import all Source Language (SL) words: 1800 Basic English
    with open("en.txt","r") as en:
        enD = en.read().split("\n")

    #Create New matrix in terms of the input SL
    #Each nth row represents the possible translations of the nth word
    for i in range(len(Words)):
        if Words[i] in enD and Matrix[enD.index(Words[i])] != '':
           Tmatrix.append(Matrix[enD.index(Words[i])].split())
    #Carry through unknown words
        else:
            Tmatrix.append([Words[i]])

    #Create permutations of possible translations of words
    #Uses SenGen function
    SMatrix = SenGen(Tmatrix)


    #Steepest Acsent Algorithm
    P = 0  #Start with probability 0 (minimum)
    for i in range(len(SMatrix)):   #For each Sentence
        if Prob(SMatrix[i])>P:      #Determine Probability
            P = Prob(SMatrix[i])    #Retain probability if higher than previous
            Trans = SMatrix[i]
    Trans = RBMT(Trans, Caps)

    return " ".join(Trans)  #Return resultant string

#Model 2: Sub Function
#Generates combinations of sentences
def SenGen(Matrix): #Takes input List of Lists (Matrix)
    #Create permutations and store to placeholder variable

    Gen = list(itertools.product(*Matrix))
    SMatrix = [] #Permutation Matrix variable

    #Called function for permutations uses tuples
    #Loop through elements, convert to list, store in Permutation Matrix variable
    for i in range(len(Gen)):
        SMatrix.append(list(Gen[i]))

    #Print number of generations
    print("Generating:", len(SMatrix), "Possible Sentences")

    #Optimise
    SMatrix = Optimse(SMatrix)

    return SMatrix #Return Permutation Matrix
    #Note does not produce arrangments therefore not true permutation



#Model 2: Sub Function
#Calculate Probability of a vector sentence
def Prob(Vec): #Takes input list of words, representing a sentence string
    #Data is finite therefore if no data is available
    #Assume probability is less than data obtained and use Fail Data


    for i in range(len(Vec)):
        if "_" in Vec[i]:
            Vec[i]=Vec[i].replace("_"," ")
    Vec =" ".join(Vec)
    Vec = Vec.split()

    ThFail = 0.00009 #Fail Probability of 3-gram (triplets of words)
    TFail = 0.00009 #Fail Probability of 2-gram  (pairs of words)
    OFail = 0.00000009 #Fail Probability of 1-gram (words)

    ##Following Data is organised such that nth
    ##elements will be corresponding per ngram
    ##E.g: The nth element of the 2gram probability
    ##list is the probability of the nth 2gram element word

    #Open 3 Gram Data store to variable
    with open("3Gram.txt","r") as f:
        TriGram = f.read().split("\n")
    #Open 3 Gram probability Data store to variable
    with open("P-3Gram.txt","r") as f:
        PTri = f.read().split()
    #Open 2 Gram Data store to variable
    with open("2Gram.txt","r") as f:
```

```
    TwoGram = f.read().split("\n")
#Open 2 Gram probability Data store to variable
with open("P-2Gram.txt","r") as f:
    PTwo = f.read().split()
#Open 1 Gram Data store to variable
with open("1Gram.txt","r") as f:
    OneGram = f.read().split()
#Open 1 Gram probability Data store to variable
with open("P-1Gram.txt","r") as f:
    POne = f.read().split()


##Decomposes input sentence into 3-gram words:
##E.g: go to the shop => (go to the, to the shop)
ThVec=[] #Vector/List for Storing 3 Grams from sentence input
#Create 3 grams of string input
for i in range(len(Vec)- 2):
    #There are N-2 possible 3-grams for N words
    #Join the elements together with " " space, to match the Data
    ThVec.append(" ".join([Vec[i],Vec[i+1],Vec[i+2]]))

##Calculate 3Gram Probability
PTh = [] #Store probability of each 3gram
for i in range(len(ThVec)):  #For each 3gram
    if ThVec[i] in TriGram:  #If probability exists store it
        PTh.append(float(PTri[TriGram.index(ThVec[i])]))
        #Expanlation:   Append the probability value of current word
        #               from probability data to Storage
        #               using the index of the current 3-gram
        #               word from position in the 2-gram data

    else:
        PTh.append(ThFail)    #Else use Failure Probability


##Decomposes input sentence into 2-gram words:
##E.g: go to shop => (go to, to shop)
TVec=[] #Vector/List for Storing 2 Grams from sentence input
#Create 2 grams of string input
for i in range(len(Vec)-1):
    #There are N-1 possible 2-grams for N words

    #Join the elements together with " " space, to match the Data
    TVec.append(" ".join([Vec[i],Vec[i+1]]))

##Calculate 2Gram Probability
PT = [] #Store probability of each 2gram
for i in range(len(TVec)):  #For each 2gram
    if TVec[i] in TwoGram:  #If probability exists store it
        PT.append(float(PTwo[TwoGram.index(TVec[i])]))
        #Expanlation:   Append the probability value of current
        #                word from probability data to Storage
        #               using the index of the current 2-gram word
        #                from position in the 2-gram data

    else:
        PT.append(TFail)    #Else use Failure Probability

##Calculate 2Gram Probability
```

```
    PO = [] #Store probability of each 1gram
    for i in range(len(Vec)):   #For each 1gram
        if Vec[i] in OneGram:   #If probability exists store it
            PO.append(float(POne[OneGram.index(Vec[i])]))
            ##Explanation: Refer to 2gram


        else:
            PO.append(OFail)    #Else use Failure Probability

    P = 1 #Let Probability = 1
    for i in range(len(PO)): #Multiply by all 1gram probability
        P = P*PO[i]
    for j in range(len(PT)): #Multiply by all 2gram probability
        P = P*PT[j]
    for j in range(len(PTh)): #Multiply by all 3gram probability
        P = P*PTh[j]

    return P #Return Resulting Probability

#============Hybrid Rule Based Functions============#

#Check Matrix of sentence Vectors for bad Sentences
def Optimse(Matrix):
    #Optimise: Contractions
    #Assumes contractions can’t be the first word
    for i in range(len(Matrix)):
        if "’" in Matrix[i][0]:
            Matrix[i] = "DELETEME"

    #Optimise: Definite Gramatical case
    #Converts all forms a case to single one that is corrected after
    #Does not effect probability as it is done to all
    The = ["’r","yr"]
    IN = ["yn","ym","yng"]

    #Check every word
    for i in range(len(Matrix)):
       for j in range(len(Matrix[i])):
    #Subrule: Definite Article
          if Matrix[i][j] in The:
              Matrix[i][j] = "y"
    #Subrule: IN cases
          if Matrix [i][j] in IN:
              Matrix[i][j] = "yn"

    #Remove dead elements
    while "DELETEME" in Matrix:
        Matrix.remove("DELETEME")

    #Removes all duplicate elements in matrix
    Matrix.sort()
    #Remove Repeats
    Matrix = list(Matrix for Matrix,_ in itertools.groupby(Matrix))
    #Declare number optimised to
    print("Optimised to:", len(Matrix), "Sentence(s)")

    #Return Optimised Matrix
    return Matrix
```

```
#Rule Based Proofer
def RBMT(Vec,Caps):

    #Gramatical Rules that need to check each word
    for i in range(len(Vec)):

    #Rule: Mutations
    #If mutation call relevant Mutation Function
        if Mutate(Vec[i]) == 2 and i< len(Vec[i])-1:
            Vec[i+1] = Trwynol(Vec[i+1])
        elif Mutate(Vec[i]) == 1 and i< len(Vec[i])-1:
            Vec[i+1] = Meddal(Vec[i+1])
        elif Mutate(Vec[i]) == 3 and i< len(Vec[i])-1:
            Vec[i+1] = Meddal(Vec[i+1])

    #Rules: Gramatatical  Cases
    #Assure Grammar for Gramatical Cases
        The = "y"
        IN = "yn"
    #Subrules: Definite Article
    #Convert all forms of Welsh "The" to 1 form
        if Vec[i] == The:
            if i == 0 or Olaf(Vec[i-1]) == 0:
                if i == len(Vec)-1 or Cyntaf(Vec[i+1]) == 0:
                    Vec[i] = "y"
                else:
                    Vec[i] = "yr"
            else:
                Vec[i] = "’r"
    #Subrules: In Cases
    #Convert all forms of Welsh "in" to 1 form
        if Vec[i] == IN:
            if i != len(Vec)-1:
                if Vec[i+1][0] == "m":
                    Vec[i] = "ym"
                elif Vec[i+1][0] == "n" and Vec[i+1][0] == "g":
                    Vec[i] = "yng"



    #Rule: Contractions
    #Assume ’ means contraction therefore combine words
    for i in range(len(Vec)):
        if Vec[i][0] =="’":
            Vec[i-1] = "".join([Vec[i-1],Vec[i]])
            Vec[i] = "DELETEME"

    #Rule: Capitalisation
    #Capitalise first letter if input is
    if Caps == 1:
        Vec[0] = Vec[0].capitalize()

    #Remove dead words
    if "DELETEME" in Vec:
        Vec.remove("DELETEME")

    #Return Proofed Sentence Vector
    return Vec
```

```
#Return 1 or 0 for Vowel or Consonant for first letter of word
#Cyntaf == First in welsh
#Used for gramatical rules: Mutations, The
def Cyntaf(Word):

    #Welsh Vowels and h
    Vowels = ["a","e","i","o","u","Y","w","h"]

    #Check if sucessive word ends in vowel or h
    if Word[0] in Vowels:
        return 1
    else:
        return 0

#Return 1 or 0 for Vowel or Consonant for Last letter of word
#Olaf == Last in welsh
#Used for gramatical rules: The
def Olaf(Word):

    #Welsh Vowels
    Vowels = ["a","e","i","o","u","w","y"]

    #Check if preceeidng word ends in a vowel
    if Word[len(Word)-1] in Vowels:
        return 1
    else:
        return 0


#Simple Mutation detection System
#Checks small sets of possible mutation causing words
def Mutate(Word):
    #Mutation word sets
    Nasal = ["yn","fy"]
    Soft = ["am","ar","at","gan","heb","i","o","dan","dros","trwy","wrth","hyd"]
    Aspirate = ["a","â", "chwe","ei","gyda","na","oni","tri","tua"]

    #Check if mutating word and return relevant
    if Word in Soft:
          return 1
    if Word in Nasal:
          return 2
    if Word in Aspirate:
        return 3
    else:
        return 0

#Soft mutation, Does not consider letters: ll or rh
#Meddal = Soft (welsh)
def Meddal(Word):
    #Mutating Consontants
    Con = ['b','c','d','g','p','t','m']
    #Mutated Consonant pair
    Mut = ['f','g','dd','','b','d','f']

    #Convert word to list to modify
    Word = list(Word)
    #If intial letter is Mutatable Consonant
    if Word[0] in Con:
    #Then mutate letter
        Word[0] = Mut[Con.index(Word[0])]

    #Convert back to word
    Word = "".join(Word)

    #Return mutated word
    return Word


#Nasal mutation
#Trwynol = Nasal (welsh)
def Trwynol(Word):
    #Mutating Consontants
    Con = ['b','c','d','g','p','t']
    #Mutated Consonant pair
    Mut = ['m','ngh','n','ng','mh','nh']

    #Mutated Consonant pair
    Word = list(Word)

    #If intial letter is Mutatable Consonant
    if Word[0] in Con:
    #Then mutate letter
        Word[0] = Mut[Con.index(Word[0])]

    #Convert back to word
    Word = "".join(Word)

    #Return mutated word
    return Word

#Aspirate
#Llaes = Aspirate (Welsh)
def Llaes(Word):
    #Mutating Consontants
    Con = ['t','p','c']
    #Mutated Consonant pair
    Mut = ['th','ph','ch']

    #Mutated Consonant pair
    Word = list(Word)

    #If intial letter is Mutatable Consonant
    if Word[0] in Con:
    #Then mutate letter
        Word[0] = Mut[Con.index(Word[0])]

    #Convert back to word
    Word = "".join(Word)

    #Return mutated word
    return Word
```