Readability Analysis of Scientific Writing

A B. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Shaurya Gomber
(160101086)

under the guidance of

Dr. Ashish Anand

to the

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Readability Analysis of Scientific Writing**" is a bonafide work of **Shaurya Gomber** (**Roll No. 160101086**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Ashish Anand**

Associate Professor,

May, 2021

Department of Computer Science & Engineering,

Guwahati.

Indian Institute of Technology Guwahati, Assam.

Abstract

Scientific writing is an integral part of academia. The research papers, journal articles, scientific posters are the major sources of information flow in the science field. The extensive research carried out by scientists and scholars and the results that they achieve are explained in these papers. The success of a paper depends on how many times it is referred or cited and number of citations has always been a metric to measure scholarly success. Studies have shown that papers which are easier to read get more attention and thus readability plays a vital role in paper's virality. This report presents a tool, **VReadA**, that visually analyzes readability of a text sample by analyzing various parameters related to linguistic complexity, predictability and coherence among sections. It can be used by the authors to get a feedback and improve on parameters which are graded poorly by the tool. This will make it easier for them to write paper in a more readable way, thus increasing the paper's chances of being viral or getting accepted in a conference or journal. This tool can also help the reviewers review papers easily as the tool outputs a pretty intuitive graphic visualization of how well the article is written and the areas of problems are highlighted.

Contents

1 Introduction **1**

1.1 Static Readability Measures 2

1.2 Machine Learning based measures 2

1.2.1 Language Modelling 3

1.2.2 Coherence Modelling 3

1.3 Organization of The Report 4

2 Related Works **5**

2.1 Static Readability Measures 5

2.2 Machine Learning based measures 11

2.2.1 Introduction to Word Embeddings 11

2.2.2 Language Modelling 15

2.2.3 Coherence Modelling 23

2.3 Readability Analysis Tool 30

3 VReadA **33**

3.1 Choice of Parameters 33

3.2 Storing and Analyzing Data 40

3.3 Visualization 42

3.4 Results 45

3.4.1 General Analysis 45

3.4.2 Perplexity Analysis 49

3.4.3 Coherence Analysis 51

4 Conclusions and Future Work **53**

References **55**

Chapter 1

Introduction

Readability is the measure of how easily the written text can be comprehended by the reader. It differs from legibility as it does not include presentation of the text, recognizability of the characters and typographic factors. Two completely legible sentences portraying the same thing can differ substantially in terms of readability. Readability is related to how easily the reader grasps what the piece of text wants to portray. This leads us to infer that texts with long sentences and difficult words would score less on readability which is indeed true as average word length, average sentence length, percentage of polysyllabic and difficult words are some of the commonly used parameters for measuring readability.

Scientific writing includes research papers and journals written by scientists and scholars to share their innovation and results with other scientists. Scientific writing is the medium of communication of ideas in the science field. The number of publications and number of citations is often used to indicate scholarly success. Intuitively, it makes sense that more readable the paper is, easier it will be for reviewers and other scientists to understand it and thus it will have high chances of being published or cited. [GPL12] studies how linguistic style affects virality of scientific abstracts and concludes that well-written abstracts get more attention and are downloaded more. [LBD+19] analyzes how the linguistic complexity of

an article affects its impact. These papers suggest that readability has a significant role to play in scientific writing. In light of this, the tool discussed in this thesis (VReadA) is developed to measure all aspects of readability and give a visually rich feedback to the author so that he/she can work on the less readable parts of the text. The subsequent sections discuss the two broad categories of readability measures.

1.1 Static Readability Measures

Static Readability Measures are the parameters to measure readability which can be calculated for the given text directly without using a machine learning model. These are various static readability measures like average word length, average sentence length, Type-to-token Ratio (TTR) etc. which judge the text for various types of syntactic and lexical complexities. Various reading ease scores like Flesch Reading Ease, Gunning Fog Index, Flesch Kincaid Grade etc. are also available that make use of the static readability measures and provide a metric to compare readability of texts. Though these measures are easier to calculate, they are limited as they only analyze a sentence structurally and do not consider whether it is meaningful or in context with the text sample. These can be achieved by the machine learning based measures discussed in the next section.

1.2 Machine Learning based measures

Because of the advancements made in Machine Learning and Deep Learning, people have started to think of a machine learning approach to everything. With the rise of social media platforms, like Twitter, Facebook etc., a lot of language data in form of tweets, comments, posts and news is available on the Internet which can be used to train the Machine Learning models. This has led to introduction of Machine Learning based methods to solve various issues in Natural Language Processing(NLP). These methods allow us to capture those hidden properties and facts about the texts that can not be computed by the

Static Readability measures discussed above. The two methods that are used in VReadA are discussed in the subsequent sections.

1.2.1 Language Modelling

Language modelling means to create a model that is able to learn and estimate the probability distributions of words and other units in a language. This way, language modelling allows us to calculate the predictability of a text sample, which can then be used to comment on the readability of text samples. Recurrent Neural Networks (RNNs) and Long Short Term Memory Networks (LSTMs) are being used widely to retain and understand context and predict the next word and these can be used to calculate the predictability of a text sample. This predictability of a text sample can be be used to indicate readability as the more predictable a text is, the less the reader has to think and can understand easily what is mentioned in the text.

1.2.2 Coherence Modelling

Machine Learning models have also been used to measure coherence between the sections in an article. Coherence of an article measures how well connected are its consecutive sections. A coherent article is well ordered with respect to its sentences and sections. It is obvious that better coherence will imply better understandability of the article, thus increasing its readability. For measuring coherence, we need to see if the neighbouring sentences have similar meanings. The meanings can be understood by a human reader but it is a difficult task for a computer. This semantic analysis is made possible the machine learning models like Word2Vec and Doc2Vec, which take in a word and doc respectively and give a vector representation of it. This vector is in the semantic space and two words are said to be portraying similar meanings, if there vectors are close-by in this semantic space. The sentence vectors of a sample can be analyzed to calculate the coherence of the sample which directly affects the readability of the sample.

1.3 Organization of The Report

This chapter touched on why readability is important in scientific writing and what are the types of readability measures. In Chapter 2, we discuss in detail the readability measures, language modelling and coherence modelling approaches. In Chapter 3, we will see how VReadA incorporates the things discussed in chapter 2 and helps us in analyzing the readability of text. It also includes the results that VReadA gives with different samples and observes that it indeed indicates and compares the readability accurately. We conclude with some future works in Chapter 4.

Chapter 2

Related Works

In subsequent sections, we will discuss what all work has been done in the field of Static Readability Measures, Language Modelling and Coherence Modelling and try to pick the important points and suggestions to incorporate in VReadA.

2.1 Static Readability Measures

[LBD+19] analyzes the relation between linguistic complexity and scientific impact. It measures readability in terms of linguistic complexity and gives a quite elaborate breakdown of the things that constitute linguistic complexity. It mentions that the linguistic complexity can be further divided into two categories:

1. Syntactic Complexity: This is basically the structural complexity of text. A very long sentence or a sentence with too many clauses is not good for readability and such type of measures come under syntactic complexity. It comprises of two things:

    - Sentence Length: This takes into account the length of the sentences while calculating readability. It can be measured by many different metrics like average sentence length, T-unit length, clause length etc. A very long sentence is not good for readability. This can thus be inferred from high value of this parameter.

- Sentence Complexity: This takes into account the complexity of the parse tree of the sentence. It can be measured by using the number of clauses per sentence, number of T-units per sentence, height of the parse tree of the sentence etc. A sentence with too many clauses can be hard to comprehend and this can be inferred from a high value of this parameter.

2. Lexical Complexity: This measures the richness of vocabulary in the text. A sentence with too many parts of speech or too many complex words can be difficult to comprehend and such type of measures come under lexical complexity. It comprises of three things:

    - Lexical Diversity: This is the measure of the number of different words used in the text. It is usually measured by counting the number of unique words in the text or by calculating the Type-Token Ratio (TTR) which is the number of unique words divided by the total number of words in the text. TTR is usually better to use as it is normalized by the length of the text and gives better metric for comparison.
    - Lexical Sophistication: It measures the sophistication of lexical items present in the text. It is usually measured by calculating the average word length and analyzing variation of word lengths in the text. A sentence with too many long complex words is hard to understand and this can be inferred from this parameter.
    - Lexical Density: It measures the density of lexical items present in the text. It is usually measured by calculating the ratio of lexical items to the total number of words in the sentence. A sentence with too many parts of speech involved can be hard to understand and this can be inferred from this parameter.

The parameters discussed in [LBD+19] measure different parts of linguistic complexity. Many formulas have been created which incorporate these parameters and give a final

readability score to the full text sample. Some of these formulas include :

1. Gunning Fox Index: The Gunning Fox index is the estimate of the number of years of schooling (formal education) that will be required by a person to understand the text. It basically takes into account how long the sentences used in the text are and how many long words (in terms of syllable count) are there in the text, to determine the complexity of the text. It is calculated as:

$$0.4 * (\text{Average Sentence Length} + \text{Percentage of Hard Words}) \tag{2.1}$$

   where:

   Average Sentence Length is calculated by dividing the total number of words in the text by the total number of sentences in the text,

   Percentage of Hard Words is the percentage of hard words present in the text. A word is considered hard if it is composed of three or more syllables (suffixes are not included in counting syllables). Note that proper nouns are not counted in hard words.

   Limitations: Though it gives good performance on hard to comprehend text samples, it is not always the case that longer words with more syllables are difficult to understand (like the word "excitement").

2. Dale-Chall Formula: The Dale-Chall formula is based on a list of 3000 easy words compiled by Dale and Chall, which they claim can be easily understood by fourth grade students. It takes the average sentence length and the percentage of hard words used in the text to determine the complexity of the text. To calculate it, first the raw score is calculated as:

$$0.1579 * \text{Difficult Words Ratio} + 0.0496 * \text{Average Sentence Length} \tag{2.2}$$

where:

Difficult Words Ratio is the number of difficult words in the text divided by the total number of words in the text. Difficult words are those which are not present in the "easy" list of words described above. (Note that the list of words only contains root words, i.e. basic forms without tenses. So, while counting easy words in the text, all the plurals, past tense forms etc. of the words in the "easy" list should be considered), Average Sentence Length is calculated by dividing the total number of words in the text by the total number of sentences in the text.

The raw score is then used to calculate the final score as:

$$\text{final score} = \begin{cases} \text{raw score} + 3.6365, & \text{if Difficult Words Ratio is } \geq 0.05 \\ \text{raw score}, & \text{otherwise} \end{cases}$$

Limitations: There can be many words which are easy to comprehend but are not in the list of 3000 "easy" words as either they are not used that frequently (hence are ignored) or are originated after the list was fixed.

3. SMOG Formula: SMOG or Simple Measure of Gobbledygook was developed by G. Harry McLaughlin. It estimates the years of education a person must have to understand the text sample. It works more accurately for long texts (more than 30 sentences) as it samples out 30 sentences to calculate the score. The SMOG formula is very accurate and easy to calculate and works very well for medical literature. It is calculated as:

$$1.043 * \sqrt{\text{Number of polysyllabic words} + (30/\text{Total Sentences})} + 3.1291 \qquad (2.3)$$

where polysyllabic words are the ones with 3 or more syllables.

McLaughlin suggested to use this on a text sample with more than 30 sentences as

follows:

- Count ten sentences from beginning of the sample, 10 from the middle and 10 from the end.
- Count the total number of polysyllabic words in these 30 sentences.
- Take the square root of the count of polysyllabic words and round it to the nearest integer.
- The final score would be 3 added to the rounded square root derived above.

Limitation: Works well for only large text samples (with more than 30 sentences).

4. Flesch-Kincaid Readability Tests: The Flesch–Kincaid readability tests use word and sentence length to determine the complexity of a text sample. The tests indicate the difficulty in comprehending the text sample.

   - Flesch Reading Ease: The reading ease test devised by Rudolf Flesch evaluates a text on the basis of the average sentence and word length present in the text and gives a score indicating how easy it is to comprehend the text sample. High scores like 90 to 100 indicate the material is very easy to understand and can be easily understood by an average 11-year old student. Low scores like 0 to 30 are very difficult to read and can be understood only be reading graduates. Polysyllabic words affect this score significantly more than they do the grade level score. It is calculated as:

     $$206.835 - 1.015 * \text{Average Sentence Length} - 84.6 * \text{Average Syllables per word} \tag{2.4}$$

     where:

     Average Sentence Length is calculated by dividing the total number of words in the text by the total number of sentences in the text,

Average Syllables per word is calculated by dividing the total number of syllables in the text by the total number of words in the text.

- Flesch-Kincaid Grade Level: It estimates the number of years of education required to understand the text. The lesser the grade level, the more readable a text will be as the less grade indicates that the text can be comprehended with less years of education. It is calculated as:

$$0.39 * \text{Average Sentence Length} + 11.8 * \text{Average Syllables per word} - 15.59 \quad (2.5)$$

where:
Average Sentence Length is calculated by dividing the total number of words in the text by the total number of sentences in the text,
Average Syllables per word is calculated by dividing the total number of syllables in the text by the total number of words in the text.

The measures and scores discussed above grade a text sample on its linguistic complexity (and thus readability) by analyzing the sample on various aspects of linguistic complexity.

Moving further, [LN13] discusses about how visual nature of an article can increase its readability. By visual nature of the article, we mean its ability to create an image in the reader's mind. It seems intuitive and logical that a reader can understand a text better if he/she can visualize it. Visual nature of an article can be measured by counting the number of visual words in it. Some words like "big", "encircled", "engulfed", "glowing" are more visual than others as they create an image in our mind. The MRC psycholinguistic database ([mrc]) is an elaborate database having imagery rating of words. This can be used to find out if a word is visual or not. The database has imagery rating 0 for non-visual words while the visual words are rated from 100 to 700 depending on their ability to invoke an

image in the reader's mind. An article with many visual words would be very readable as it would enable the reader to easily visualize what is written. This suggests that proportion of visual words in an article can also be taken while measuring readability.

2.2 Machine Learning based measures

In this section, we will discuss about the readability measures that require the training of a machine learning model for computation. We start this section by an introduction to word embeddings, which are basically the vectors in the semantic space assigned to a word. These embeddings are widely used in the machine learning based approaches used in NLP to represent the semantic information encoded in the word. After discussing about embeddings, we will see the works done in Language Modelling and Coherence Modelling.

2.2.1 Introduction to Word Embeddings

Word embeddings are learned representations for text. It maps words to basically a vector of some dimensions in such a way that the words with similar meanings are given similar representations. That is, words with similar meaning are closer to each other in this vector space (the semantic space as we can call it). Just for understanding the feel behind the embeddings, the various numbers in the vector representation of a word can be thought of as features of the word. For eg, there could be a gender feature which which can have high value for female gender words like queen, woman etc. and less value for male gender words like man, king etc (This is just to give a sense of how word embeddings are able to capture the semantics of a word. The features learnt are more complex than just "gender"). The effectivity of the word embeddings can be seen from the fact that if we take the vectors of words "King", "Man" and "Woman" from a well-trained embedding model and perform the following vector operations $\overrightarrow{King}$ - $\overrightarrow{Man}$ + $\overrightarrow{Woman}$, then we will get a vector which will be very close to the vector of the word "Queen". This shows how effectively the embeddings retain the semantics of the word. Subtracting the vector of the word "man" from the vector

of the word "king" and adding the vector of the word "woman" to it effectively changed the gender in the vector space and gave the vector of queen. This optimal retention of the semantics when converted to vectors is what makes the use of embeddings correct and so popular.

One such model that very effectively finds out the word embeddings and is quite popular is the Google's word2vec ([MSC+13]). We will now discuss in brief the main idea used by it to get the embeddings (we omit the mathematical details for simplicity, these can be looked in the paper). ([MSC+13]) discusses two ways for creating the embedding which are both very effective and are suited to different condtions:

1. Continous Bag of Words (CBOW): This technique tries to predict or guess a word using its neighbouring context words. The words are initialized with one hot encoding vectors which are vectors of length V(size of vocabulary) with only one position having 1 and rest being 0s. This position is the index of the word in the vocabulary. Fig. 2.1 shows the very basic architecture of the CBOW model.

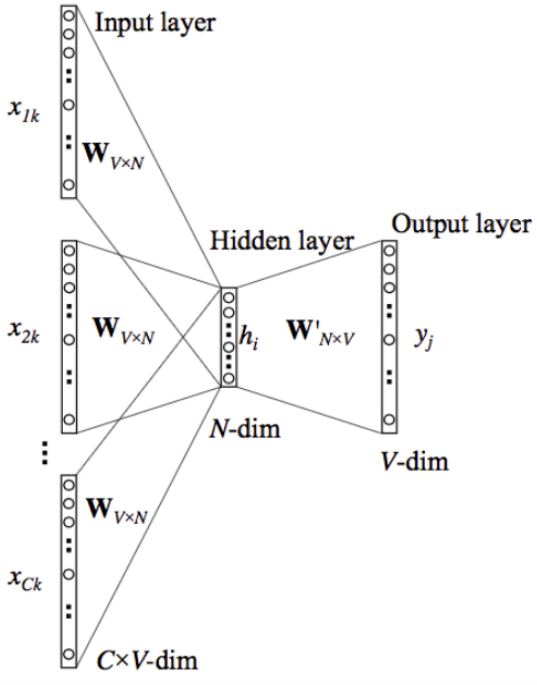

Figure 2.1: CBOW model. Image Source: [w2v]

If say we need to get projection vectors of dimension N, then a weight matrix W (of size VXN) is initialized randomly and is used to get the N dimensional projections of the C context vectors (x_{1k},x_{2k} .. x_{Ck}). The projections of these context vectors are averaged element-wise at the hidden layer to get a single n dimensional vector, which is the output of the hidden layer. This output is multiplied by $W^{'}$ (of size NXV, also initialized randomly) and softmax activation is applied at the last layer thus converting the v-dimensional vector to a probability distribution over v values. While training, we will have the j_{th} vector with only one 1 at the j_{th} index. This can be used with the output probability distribution to find out the cross entropy loss (used widely in logistic regression). This loss can be used to update the weights using the Backpropogation algorithm.

Once the training is done, we can obtain the projection of a word by multiplying its one hot encoding with the matrix W.

2. Skip-Gram: It is different from CBOW in the sense that it tries to predict the context words from a given word. Fig. 2.15 shows the very basic architecture of the Skip-Gram model.

   If say we need to get projection vectors of dimension N, then a weight matrix W (of size VXN) is initialized randomly and is used to get the N dimensional projection of the vector of our center word x. This N dimensional output of the hidden layer is multiplied by $W^{'}$ (of size NXV, also initialized randomly) and softmax activation is applied at the last layer thus converting the v-dimensional vector to a probability distribution over v values. While training, for every such x, say we have C vectors of the context words (y_1,y_2 .. y_C). For each of these vectors, we find out the the cross entropy loss using the final probability distribution obtained after softmax. The total loss is the sum of these losses for all the c context vectors. This loss can be used to update the weights using the Backpropogation algorithm.

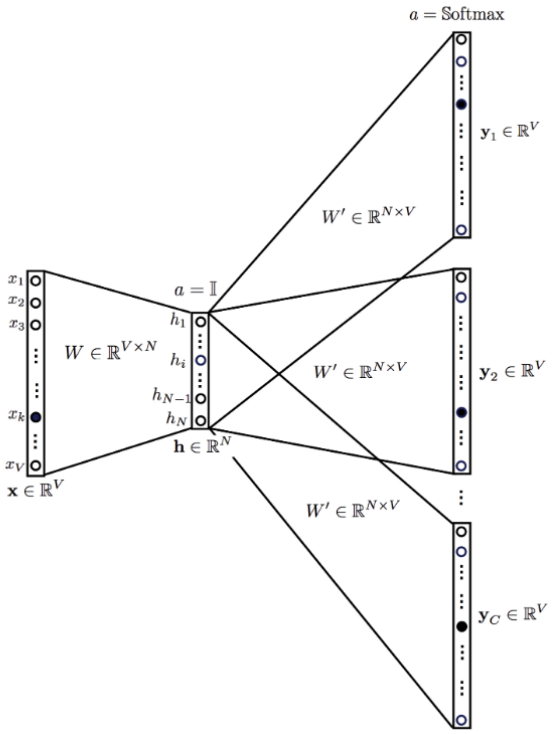

Figure 2.2: Skip-Gram model. Image Source: [w2v]

Once the training is done, we can obtain the projection of a word by multiplying its one hot encoding with the matrix W.

While Skip-gram works well with small amounts of training data and represents the rare words well, the CBOW model is several times faster to train than the Skip-gram model and gives better results for frequent words.

Many more embedding models like GloVe ([PSM14]) by Stanford and fastText ([BGJM16]) by Facebook are also present and have been found very efficient at retaining the semantics of a word in the vector form. All of these models provide pre-trained embeddings as well. We can either use these directly or can create our own word embeddings trained on our corpus using the ideas discussed above.

Sometimes, we want embeddings for a sentence or a piece of text. One approach could be averaging the vectors of individual words present in the text. But this does not take into consideration the order of words in the sentence as if the order is changed also, the average of

vectors would remain the same. The order of a sentence is quite crucial for the semantics of it. Google's doc2Vec [LM14] explains how we can get the sentences and document embedding by tweaking the word2vec model explained above. word2vec ([MSC+13]) and doc2vec [LM14] are very efficient in retaining the semantic information and thus are used in many machine learning methods used to solve various NLP problems.

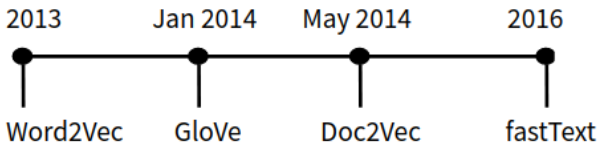

Figure 2.3: Word Embeddings Literature Evolution

In the subsequent sections, we will be discussing Language Modelling and Coherence Modelling which makes use of these embeddings.

2.2.2 Language Modelling

The readability measures and scores discussed till now have measured the linguistic complexity of the text. But as discussed eariler, predicatability of a text can also be thought of as a viable metric of readability. The intuition behind this is that if while reading, a reader is able to predict what comes next, it means the article is well-written and the reader is understanding what the article tries to state. This suggests that we can make use of predictability as a measure of readability. Finding predictability of a text is not that straight-forward and here comes language model in the picture. A language model is a model that can learn and estimate the probability distributions of words and other units in a language. The language model thus can help us predict the next word given a partial sentence based on what it learns from the corpus on which it is trained. The language modelling can further be broadly divided into 2 categories:

Statistical Language Modelling

Statistical Language Modelling ([Col13], Chapter 3 of [JM00]) works by assigning probabilities to sentences and sequences of words. A sequence of n words is called a n-gram. So, a 2-gram or bigram is a sequence of two words like "My name" and a 3-gram or trigram is a sequence of three words like "My name is".

We will now see how a trigram language model can be generated, given we have a corpus of text to train on. We process whole data present in the corpus and maintain the count of all appearing bigrams and trigrams in the corpus. Say b(u,v) is the number of times the bigram (u,v) appears in the corpus and t(u,v,w) be the count of the number of times the trigram (u,v,w) appears in the corpus. Once we are done with this, we are ready to find predictability of the text. For this, first we state that now we are in a position to calculate p(w|u,v) i.e the probability that w appears after the bigram (u,v). This can be calculated as:

$$p(w|u,v) = \frac{c(u,v,w)}{b(u,v)} \tag{2.6}$$

This tells that the probability that w will appear after (u,v) is the ratio of the number of times the trigram (u,v,w) appears in the corpus to the number of times the bigram (u,v) appears in the corpus. This seems right intuitively as we are just seeing that in our corpus, whenever we had the bigram (u,v), how many of those times was the next word w by counting the trigrams (u,v,w). As we are using that corpus statistics (count of bigrams and trigrams) to learn and estimate the probability distribution, this comes under the statistical way of modelling. Thus, given a sentence say x_1, x_2 ... x_n (where x_i is the i^{th} word of the sentence), its probability under the trigram model is given by

$$P(x_1, x_2, ...x_n) = \prod_{i=1}^{n} p(x_i|x_{i-1}, x_{i-2}) \tag{2.7}$$

where p($x_i|x_{i-1}$,x_{i-2}) can be calculated using 2.6.

If we have a test set of sentences say s_1, s_2 ... s_n (where s_i is the i^{th} sentence), we calculate our model's performance on this test set by calculating the perplexity. To calculate perplexity, we first calculate the l value as follows :

$$l = \frac{1}{N} \sum_{i=1}^{n} P(s_i) \tag{2.8}$$

where N is the total number of words present in all test sentences and P(s_i) can be calculated using eq. 2.7.

Now, the perplexity of our model as measured on the test set is defined as

$$2^{-l} \tag{2.9}$$

where l is given by equation 2.8.

The lower the value of perplexity, the better our model does on the test set because that implies a greater l value from eq. 2.9 and greater l implies greater probability of the sentences from the eq. 2.8. Once the model is trained, we can find the perplexity of any text sample by the method described above. Low value of perplexity means that sample is predictable and thus easily understandable by the reader as the reader can easily predict the next word while reading. The perplexity values can be used to compare performances of different statistical models.

This statistical approach has one major drawback. Let us assume we have a sentence "The dog is eating." and we use our trigram model to find its probability. If the trigram (the,dog,is) is not present in the corpus, the probability assigned to the sentence will be 0 as p(is|the,dog) will be 0 (Refer equations 2.6 and 2.7). And if say there is no bigram (the,dog), the estimate will not be well defined. This can be taken care of using smoothing methods like linear interpolation and discounting as explained in [Col13].

[CTC04] explains another statistical modelling approach based on multinomial naive bayes classification to estimate the grade level of a text sample.

The statistical methods have a disadvantage that they do not generalize well. This means that even if the corpus had a statement "The cat is eating.", still the model cannot generalize it to a dog and predict "The dog is eating." even though the dog is similar to cat in the context of the sentence. The statistical methods can only predict words which they have seen and can't make such logical generalizations. These logical generalizations need semantic understanding which can be achieved by the neural language modelling techniques discussed in the next secion.

Neural Language Modelling

Neural language models (Chapter 7 of [JM00]) or neural network based language models are models that use neural network to learn and predict distribution of words and linguistic units. These models don't require smoothing, can learn and analyze longer histories and generalize pretty well. The only drawback they have is it takes a lot of time to train a neural language model as compared to the traditional statistical language model.

In neural networks, a word is represented by its word embedding. This representation of the prior context as embeddings rather than exact words (as in case of statistical modelling) is what enables the neural language model to generalize on unseen test data better than statistical (n-gram) language models. For eg. consider a sentence "I will feed my cat." which we encounter in our corpus (It contains statements about cat only). Now if we go on to predict what will come after "I will feed my" using the trigram model discussed above, it predicts cat with very high probability and gives 0 probability to dog whereas a neural language model will give high probabilities to both the dog and the cat as the embeddings of cat and dog are similar.

So, while training our neural model, we give as input the embeddings of the prior words (context) and the embedding of next word to our model so that it trains and learns how

to predict the next word (basically it's embedding) given the embeddings of prior words. The word embeddings can be derived in two ways :

- A pre-trained model based on an embedding finding algorithm could be used diretcly to get the embeddings. As discussed earlier, word2vec ([MSC+13]), gloVe ([PSM14]) and fastText ([BGJM16]) are the most widely used models to get word embeddings.
- We can set the embeddings of the words as variables to be learned and can modify our loss function so as to learn the embeddings of the word while we are predicting the next word. Chapter 7 of [JM00] explains in detail how this can be used to achieve the desired results.

Language has a sequential structure to it. The sentences we utter are sequences of words and every word uttered depends on prior words (or context). Traditional neural networks have this short-coming that they can't keep track of the context properly. This is where Recurrent Neural Networks (RNNs) come into the picture. They are networks with loops. This allows context information to persist and travel along the sequence. Figure 2.4 shows a basic RNN cell. Figure 2.5 visualizes a RNN cell by unrolling it.

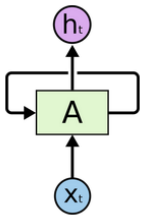

Figure 2.4: Loop in Recurrent Neural Network. Image Source: [Ola15a]

As can be inferred from the unrolled network, input x_i as well as context output from the previous layer is used to find the prediction at this time (h_i) as well as the context output that passes to the next layer. This passing of context among layers is what makes RNN different from traditional neural networks and help RNN learn and predict sequences easily.

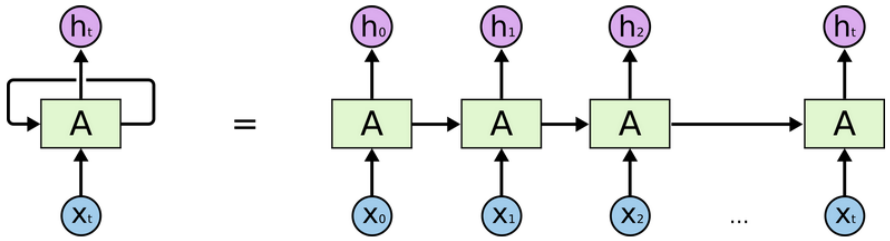

Figure 2.5: Unrolled Recurrent Neural Network. Image Source: [Ola15b]

Though, when we have very large sequences, context information at a layer will have very small contributions from the inputs which were long before this layer. This can be seen in figure 2.6 where h_{t+1} will have very less contributions from x_0 and x_1. This happens because their contributions come while passing through many layers and at each layer, the contribution can be multiplied by a weight (<1) which could vanish the effect of x_0 and x_1. This is called the "vanishing gradient" problem.

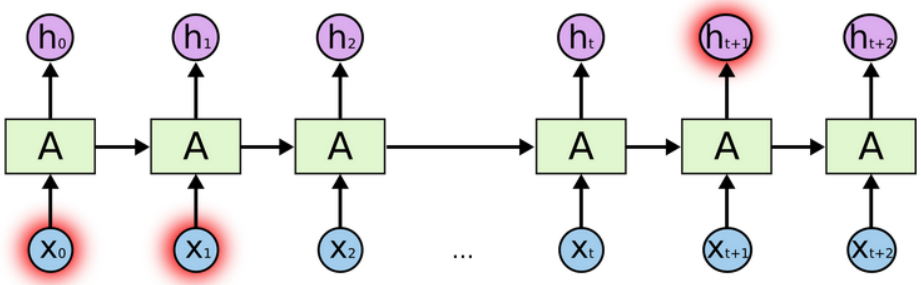

Figure 2.6: Long term dependency issue in RNN. Image Source: [Ola15c]

To tackle this problem, Long Short-Term Memory(LSTM), which is a type of artificial RNN, is used. A LSTM unit comprises of an input gate, an output gate and a forget gate. The input gate controls contributions of the input (x_i) and the forget gate controls what part of context information that comes from the previous cell will be used. So context information to be passed to the next cell will be controlled input, which is given by the input gate plus the part of context from the previous cell that this cell does not forget. The output gate will then use this context information to predict an output. These 3 gates basically control the information flow in LSTMs.

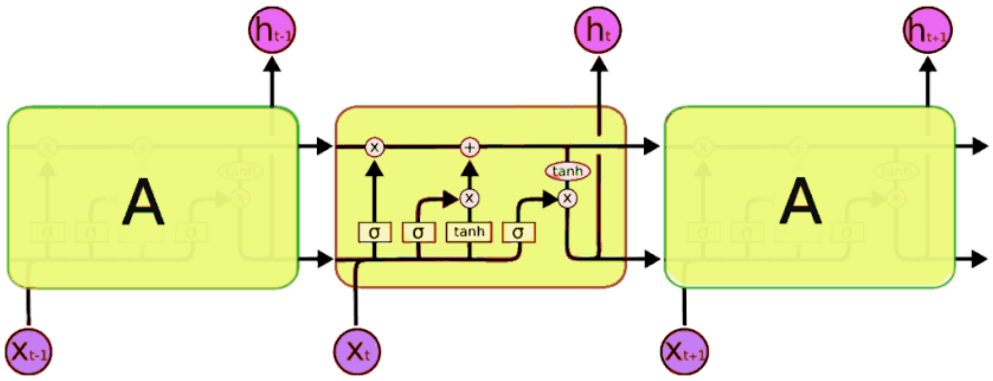

Figure 2.7: Long Short-Term Memory Networks. Image Source: [Ola15d]

Most of the neural language models use deep LSTM networks (deep means many layers of LSTM stacked together) for language modelling as LSTMs can learn context easily and do not suffer from the vanishing gradient problem. [JVS+16] describes in detail how the LSTMs can be used to create very efficient language models.

While using deep neural networks, one of the major problems is over-fitting which can occur if we train a large network on a relatively small training set. This happens because while training, neurons develop co-dependency among each other. That is, some neurons start depending on each other while training and this hampers their ability to learn robust features. To tackle this co-dependency issue, [HSK+12] proposes the technique of Dropout while training large neural networks. In this technique, while training, for each training sample, p fraction of units in every hidden layer are ignored(or zeroed out). While testing, we use all the nodes but first we divide all the weights by p to compensate for the fact that while training at each step, p fraction of units were not considered. As, while training, different sets of neurons are used at each step, any two neurons can not just depend on each other and this enables the neurons and the network to learn more robust features.

But dropout can not be directly applied to RNN and LSTM networks as in these networks, previous state is fed back into the network which allows them to retain context and understand dependency. Applying dropout hinders the ability of LSTM networks to retain long term dependencies. [WZZ+13] proposes the DropConnect regularization

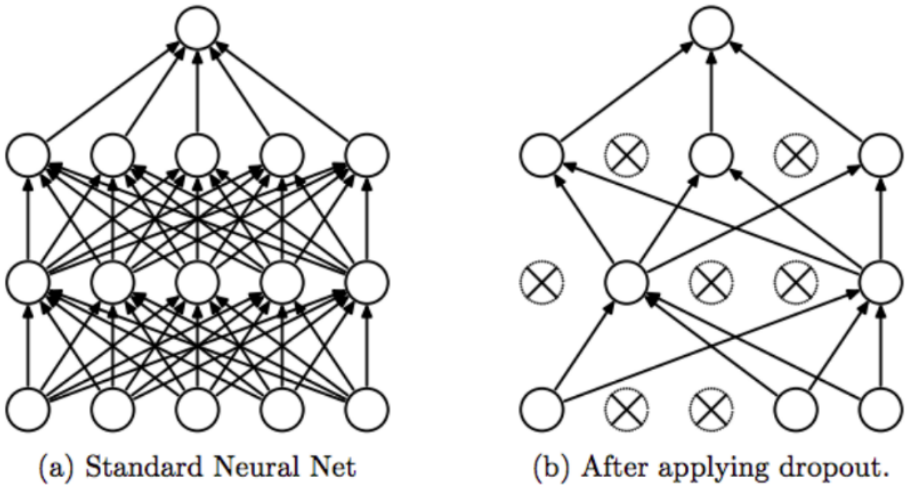

Figure 2.8: Image Source: [SHK+14]

method that is similar to Dropout but in this, instead of the nuerons, p fraction of weights are ignored(set to 0) while training. Effectively, both the Dropout and the DropConnect lets us train several models at the same time and while testing, we can just take the average of all the models. In this sense, DropConnect is better as it produces even more models as almost always, there are more connections than the units. Also, as the DropConnect does not drop units while training and only sets some weights to 0, the context is not lost and so it can be used with RNNs and LSTMs as well.

[MKS17] proposes using the AWD-LSTM model which is a deep LSTM network for Language Modelling that uses DropConnect and gives more accurate perplexity results than the previous implementations. In 2018, Google came up with BERT (Bidirectional Encoder Representations from Transformers) ([DCLT18]). It uses the Transformer model architecture instead of LSTMs. It is bidirectional in the sense that while training, it looks the text sequences both from left-to-right and right-to-left, thus enabling it to have a deeper sense of language context. It is a complex model with a large number of parameters and gives better results than the LSTM based approaches. Recently, Microsoft announced the Turing-NLG [tnl20] which is also a Transformer based model and has about 17 billion parameters. It claims to outperform the state-of-the-art language models on almost all

NLP tasks.

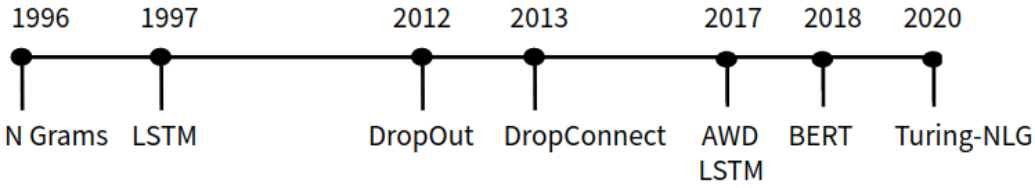

Figure 2.9: Language Modelling Literature Evolution

Having looked at the works in the field of Language Modelling, we will now discuss the developments in the field of Coherence Modelling.

2.2.3 Coherence Modelling

Coherence refers to the semantic and grammatical interconnectedness between the sentences and sections of a text sample. It measures how well an article is structured with respect to ordering of the sentences and the paragraphs. A text sample which has sentences on similar topic together and which is structured in a clear manner will be better for the reader to grasp what the text wants to portray. This can be measured by analyzing coherence in a text sample. There are two types of coherences:

1. Local coherence: This captures the relatedness in text at the level of sentence to sentence transitions. It focuses on whether the nearby sentences are related to the same topic and ordered accordingly.

2. Global coherence: It focuses on the text as a whole. It measures how the topics flow through consecutive sections and paragraphs.

[LB05] introduced the concepts of Entity Grids to measure local coherence of a text sample. Entity Grids are two-dimensional arrays that are used to store the distribution of entities across the sentences of a text sample. The entities are usually the coreferent noun phrases that are mentioned in the sample (coreference occurs when more than one noun phrases refer to the same thing, these are taken as one for the entity grid formation).

The columns have the different entities while the rows signify sentences. $(i, j)^{th}$ cell of the entity grid depicts the role of j^{th} entity in the i^{th} sentence. If it occurs as a subject in the i^{th} sentence, the cell has 's'. If it occurs as object in the i^{th} sentence, the cell has 'o'. If it is present as neither the subject nor the object, the cell has 'x'. The cell has '-' if the j^{th} entity is not present in the i^{th} sentence. Fig 2.11 shows an example of how the entity grid looks.

[LB05] treats all columns of the grid as separate Markov chains and calculates the probability of the column sequences independently for each of the entities. The coherence score can be then be calculated by multiplying such probabilities for all the entities(columns).

1 [The Justice Department]$_s$ is conducting an [anti-trust trial]$_o$ against [Microsoft Corp.]$_x$ with [evidence]$_x$ that [the company]$_s$ is increasingly attempting to crush [competitors]$_o$.
2 [Microsoft]$_o$ is accused of trying to forcefully buy into [markets]$_x$ where [its own products]$_s$ are not competitive enough to unseat [established brands]$_o$.
3 [The case]$_s$ revolves around [evidence]$_o$ of [Microsoft]$_s$ aggressively pressuring [Netscape]$_o$ into merging [browser software]$_o$.
4 [Microsoft]$_s$ claims [its tactics]$_s$ are commonplace and good economically.
5 [The government]$_s$ may file [a civil suit]$_o$ ruling that [conspiracy]$_s$ to curb [competition]$_o$ through [collusion]$_x$ is [a violation of the Sherman Act]$_o$.
6 [Microsoft]$_s$ continues to show [increased earnings]$_o$ despite [the trial]$_x$.

Figure 2.10: Sample text. Image Source: [BL08]

	Department	Trial	Microsoft	Evidence	Competitors	Markets	Products	Brands	Case	Netscape	Software	Tactics	Government	Suit	Earnings	
1	s	o	s	x	o	–	–	–	–	–	–	–	–	–	–	1
2	–	–	o	–	–	x	s	o	–	–	–	–	–	–	–	2
3	–	–	s	o	–	–	–	–	s	o	o	–	–	–	–	3
4	–	–	s	–	–	–	–	–	–	–	–	s	–	–	–	4
5	–	–	–	–	–	–	–	–	–	–	–	–	s	o	–	5
6	–	x	s	–	–	–	–	–	–	–	–	–	–	–	o	6

Figure 2.11: Entity Grid for the sample text in Fig. 2.10. Image Source: [BL08]

[BL08] is authored by the authors of [LB05] and proposes a more accurate use of entity grid for coherence modelling. It proposes that once a grid is created from the text, a feature vector of the text can be created using this grid. The different values in the feature vector will be the probability of different transitions in the grid. For eg. say the vector's first component has the probability of the transition s $\rightarrow$ o, then this can be computed for our

example grid in fig. 2.11 as follows:

The number of transitions from s $\rightarrow$ o = 1 (Microsoft column, first and second entry)

Total number of transitions = 75 (5 transitions each for the 15 entities).

So, the first position of feature vector for this text will have 1/75.

Using this method, feature vectors can be created for all the text samples. Now, machine learning techniques like SVM can be applied using these features to get the appropriate weights which when multiplied by these features give the coherence score. For eg., if we have a set of N training examples (x_i,y_i) where we know that the texts corresponding to features (x_1,x_2 .. x_N) are more coherent than their y counterparts, then our modelling objective is to find a vector w such that

$$\forall i \in 1, 2, 3..n : w \cdot x_i > w \cdot y_i \tag{2.10}$$

which boils down to

$$\forall i \in 1, 2, 3..n : w \cdot (x_i - y_i) > 0 \tag{2.11}$$

The problem in eq. 2.11 is a typical Support Vector Machine constraint optimization problem. Once the training is done, feature vectors of two texts can be multiplied by w and the resulting values can be compared to find out which is more coherent. This shows how feature vectors can be used to model coherence. This technique can be used in many tasks as document discrimination, summary coherence rating etc. [FLH14] proposes that the results can be improved further by modifying the entity grids to store the discourse role of entities in the grid rather than their role in the sentence (subject, object or x). Discourse role of entities include conjunction (when 2 entities are connected by and), contrast(when 2 entities are connected by but) etc. Filling this information in the entity grid matrix rather than 's', 'o' and 'x' helps to capture the semantic relations more accurately and give better results.

[BL08] and [FLH14] give good results but are supervised learning methods. They require

labelled texts on which they can be trained to produce the results. This makes these methods suffer from data sparsity and computational complexity. This was resolved by [GS13] which proposes a very unique and efficient graph-based method for modelling local coherence. [GS13] defines a projection graph as a graph where the nodes represent the sentences of the text sample. The graph is a directed graph and edges from i^{th} sentence to the j^{th} sentence are allowed only if i < j. That is, the edges are always forward if we see with respect to the order of the text. There is an edge between 2 sentences if they share at least one entity. There can be three kinds of graphs depending on how the edges are weighted:

1. P_U: Every edge is given weight 1.

2. P_W: The weight of edge between the sentences s_i and s_j are the number of entities shared by the two sentences.

3. P_{Acc}: It takes into account the syntactic information while giving weights to the edges. First, we define v(e,i) as the value given to entity i with respect to the i^{th} sentence. It is defined as follows:

$$\text{v(e,i)} = \begin{cases} 0, & \text{if e is not present in the } i^{th} \text{ sentence} \\ 3, & \text{if e is present as subject in the } i^{th} \text{ sentence} \\ 2, & \text{if e is present as object in the } i^{th} \text{ sentence} \\ 1, & \text{if e is present as neither the subject nor the object in the } i^{th} \text{ sentence} \end{cases}$$

Now, if E is the set of entities shared between the i^{th} and the j^{th} sentence, then the weight W_{ij} is given by:

$$W_{ij} = \sum_{e \in E} v(e,i) \cdot v(e,j) \tag{2.12}$$

Once a projection graph is created using any of the techniques above, the distance between two sentences can also be incorporated to decrease the importance of those links

which are between sentences wide apart in the text. This can be done by diving W_{ij} by absolute value of $i - j$ which indicates the distance between the sentences. The coherence of a text sample T can now be computed using the projection graph by the formula:

$$LocalCoherence(T) = \frac{1}{N} \sum_{i=1..N} OutDegree(s_i) \tag{2.13}$$

where

N is the number of sentences in the text and

OutDegree(s_i) is the sum of weights of edges leaving the node s_i in the graph.

This unsupervised approach proposed by [GS13] gives results comparable to the Entity Grid approach and has 2 advantages over it. First, the constructed projection has all the details about the entity transactions encoded and so this method needs no training. Second, it is unsupervised and needs no training and so it does not suffer from computational complexity and data sparsity issues.

The methods discussed till now just consider coherence due to local cohesion, i.e. repeated mentions of same entities. But a sample can be coherent without it sharing the same entities. This can happen because of the use of different words portraying the same meaning. [PT17] proposes a solution to this problem by taking care of the related yet not identical entities. This is done by creating a directed graph with nodes as the sentences of the text. The edges and weights are decided by measuring the semantic similarity between the sentences and thus this is called the Semantic Similarity Graph. The sentences are represented by vectors (sentence embeddings) for measuring the semantic similarity. The sentence vectors are formed by taking the average of the consisting word vectors. Pretrained GloVe ([PSM14]) vectors are used to get the word vectors. [PT17] proposes three ways of creating the graph:

1. Preceding Adjacent Vertex (PAV): While reading a text sample, if we do not understand a sentence, we usually look backwards to see what we have missed. This

intuition is used in the graph. Here, for every sentence, we will traverse backwards and find the first sentence having positive similarity with it. The similarity simi(s_i, s_j) between the sentences s_i and s_j is given by:

$$simi(s_i, s_j) = \alpha * overlap(s_i, s_j) + (1 - \alpha) * cos(\vec{s_i}, \vec{s_j}) \tag{2.14}$$

where

overlap(s_i,s_j) is the number of terms common to both the sentences divided by the total number of unique terms in the 2 sentences,

cos($\vec{s_i}$,$\vec{s_j}$) is the cosine similarity between the 2 sentence vectors,

α lies between 0 and 1 and is the balance factor.

For each sentence i, we start from $i - 1$ and go backward and stop at the first index j such that simi(s_i,s_j) > 0. An edge is created from s_i to s_j with weight simi(s_i,s_j). This means it is connected to the closest sentence behind it which has a positive similarity with it. Fig 2.12 shows a PAV graph.

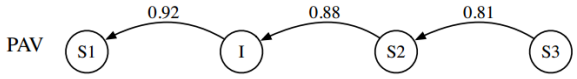

Figure 2.12: Example of a PAV graph. Image Source: [PT17]

2. Single Similar Vertex(SSV): Ideally, in a coherent text sample, we want that for a sentence i, the sentence most similar to it, say j, should be close to it as these two sentences are semantically similar and thus dependent on each other. This intuition is used in the graph. For each sentence i, we search the sentence most similar to it where similarity is the cosine similarity. The i^{th} sentence is then connected to the sentence most similar to it, say j. The weight w(s_i, s_j) between the sentences s_i and s_j is given by:

$$w(s_i, s_j) = \frac{cos(\vec{s_i}, \vec{s_j})}{|i - j|} \tag{2.15}$$

where

$\cos(\vec{s_i},\vec{s_j})$ is the cosine similarity between the 2 sentence vectors.

The distance between the sentences is taken into account while calculating weight as we will prefer the most similar sentence to be closer. Fig 2.13 shows a SSV graph.

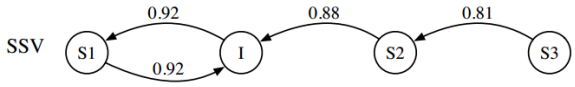

Figure 2.13: Example of a SSV graph. Image Source: [PT17]

3. Multiple Similar Vertex(MSV): It is similar to SSV with just a small modification. Here, we have a parameter θ and we connect sentence s_i to all sentences whose cosine similarities with it are more than θ. The weights to be given to these edges are same as in eq. 2.15, i.e. here also we prefer the semantically similar sentences to be closer. Fig 2.14 shows a MSV graph.

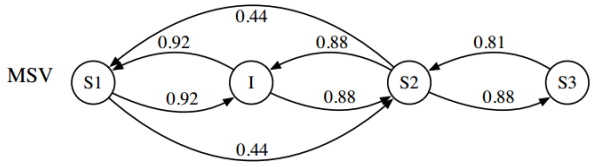

Figure 2.14: Example of a MSV graph. Image Source: [PT17]

Once the graph is created using any of the methods discussed above, the coherence of a text sample can be calculated by the following formula:

$$coherence = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{L_i}\sum_{k=1}^{L_i} weight(e_{ik}) \tag{2.16}$$

where,

N is the total number of sentences,

L_i is the number of outgoing edges from vertex i,

e_{ik} is the k^{th} edge originating from the i^{th} vertex.

The Semantic Similarity Graph has two advantages over the Entity Graph. First, it allows for edges in both the directions while Entity Graph only allowed an edge to the sentence which is forward in the text order. This allows the graph to capture the semantic dependencies on sentences that are before the sentence. Secondly, it uses semantic similarity using word embeddings rather than just relying on the common entities. This lets it to take into consideration the related yet not identical entities. This enables it to outperform both the Entity Grid and Entity Graph methods.

These were some of the major works in Coherence Modelling. Other notable works include [BL04] that uses Hidden Markov Models to observe how the topic of the content changes across sections and captures the global coherence, [EAC07] proposes a method to incorporate both the local and global coherence while calculating coherence and [CLZZ17] discusses a deep convolutional neural network that uses sentence vectors and similarities between sentences as features to calculate the text coherence.

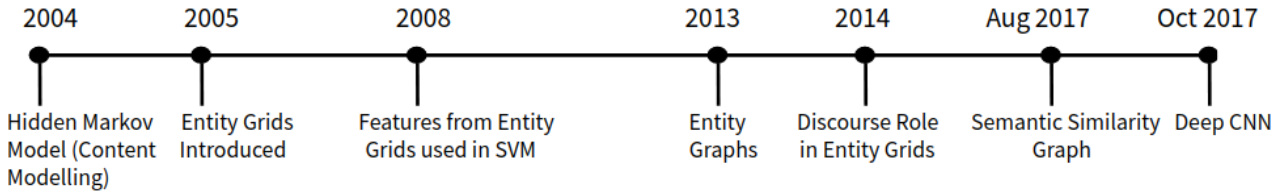

Figure 2.15: Coherence Modelling Literature Evolution

2.3 Readability Analysis Tool

[OSSK10] discusses a tool for Visual Readability Analysis (VisRA) and discusses how a tool that visualizes the readability analysis of a text sample can help the authors easily detect the parts which are linguistically complex and thus hard to understand. VisRA analyzes the readability in terms of some fixed parameters of linguistic complexity like word length, sentence length, sentence structure and gives as output a visualization of the analysis. The visualization makes it easier for the author to detect the problematic parts in the paper and the author can these correct those parts in the subsequent drafts. The tool gives different

options like corpus view, block view and detail view. Corpus view analyzes the paper as a whole, block view splits the paper into sections and analyzes each section separately while the detail view analyzes each sentence of the paper separately. This suggests that such a tool which gives a visualization of the readability analysis can help the authors increase readability of their writings.

Chapter 3

VReadA

Based on our study of related works, we create a tool, VReadA (**V**isual **Read**ability **A**nalyzer) that analyzes readability of a text sample and visualizes the analysis using a heatmap which makes it easier for authors to detect parts which are graded poorly on readability by VReadA. This will help them in revising their drafts and making their writing more reabable.

VReadA gives a sentence level readability analysis where each sentence is analyzed on the basis of chosen parameters. Then, a heatmap of the size of (Number of sentences) X (Number of features) is used to depict readability of our text sample. The color of the (i,j) cell will determine how the i^{th} sentence of our sample fares on the j^{th} parameter. VReadA's design is explained in the subsequent sections.

3.1 Choice of Parameters

VReadA rigorously analyzes the sentences in the text sample on 13 parameters. These are divided into the following four types:

1. Syntactic Complexity Parameters: These take into account the structural complexity of the text sample. A very long sentence or a sentence with too many clauses is not good for readability. Such sentences can be detected by these parameters. There are

two parameters in this category:

(a) Sentence length (SL): It is the number of words present in the sentence. It measures the sentence length component of syntactic complexity which in turn affects linguistic complexity. Low value of this parameter is good for readability as smaller sentences can be easy to comprehend.

(b) Parse Tree Height (PTH): It is the height of the parse tree of the sentence. A parse tree is basically a pictorial representation of the grammatical structure of a sentence. It measures the sentence complexity component of syntactic complexity which in turn affects linguistic complexity. Low value of this parameter is good for readability as larger height of the parse tree means that the sentence is grammatically complex and can thus be hard to interpret.

2. Lexical Complexity Parameters: These take into account the lexical complexity of the text sample which depends on the richness of the vocabulary used. A sentence with too many parts of speech or too many complex words can be difficult to comprehend. Such sentences can be detected by these parameters. There are six parameters in this category:

(a) Type-Token Ratio (TTR): It is the number of unique words divided by the total number of words in the text. It measures the lexical diversity component of lexical complexity which in turn affects linguistic complexity. Low value of this parameter means less unique words to understand, which is good for readability.

(b) Average Word Length (AWL): It is the average number of letters in the words of the sentence. It can be calculated by dividing total number of letters in the sentence by total number of words in the sentence. It measures the lexical sophistication component of lexical complexity which in turn affects linguistic complexity. Low value of this parameter means easy small words are used to write the sentence, which is good for readability.

(c) Average Syllable Count (ASC): It is the average number of syllables in the words of the sentence. It can be calculated by dividing total number of syllables in the sentence by total number of words in the sentence. It measures the lexical sophistication component of lexical complexity which in turn affects linguistic complexity. Low value of this parameter means easy small words are used to write the sentence, which is good for readability.

(d) Difficult Word Ratio (DWR): It is calculated by dividing the total number of difficult words in the sentence by total number of words in the sentence. Difficult words are those which are not present in the easy words list compiled by Dale and Chall. It measures the lexical sophistication component of lexical complexity which in turn affects linguistic complexity. Low value of this parameter means easy words are used to write the sentence, which is good for readability.

(e) Part Of Speech Ratio (POS): It is calculated by dividing the count of different types of parts of speech present in the sentence by the total number of words in the sentence. It measures the lexical density component of lexical complexity which in turn affects linguistic complexity. Low value of this parameter means that the sentence is written in an easy language without using many parts of speech, which is good for readability.

(f) Visual Words Ratio (VWR): As suggested by [LN13], this parameter calculates the ratio of visual words present in the sentence to the total number of words in the sentence. MRC pyscholinguistic database ([mrc]) is used to get imagery ratings of words present in the sentence. High value of the parameter indicates that the sentence successfully invokes an image in the reader's mind and this is good for readability.

3. General Readability Indices: These indices use the above described parameters and give a final readability score. There are 3 parameters in this category:

(a) Gunning Fog Index (GFI): As discussed before, it estimates of the number of years of schooling (formal education) that will be required by a person to understand the text (sentence here). It can be calculated by using the equation 2.1. Low value of this parameter is good for readability.

(b) Flesch Kincaid Grade Level (FKG): As discussed before, it also estimates the number of years of education required to understand the text (sentence here). It can be calculated by using the equation 2.5. Low value of this parameter is good for readability.

(c) Flesch Reading Ease (FRE): As discussed before, it estimates how easy it is to understand the text (sentence here). It can be calculated by using the equation 2.4. High value of this parameter indicates that the text is easy to understand, which is good for readability.

4. Machine Learning based Parameters: The parameters in this category are the ones obtained with the help of machine learning models and provide details that can not be extracted by the static readability measures discussed above. There are two parameters in this category:

(a) Perplexity(PRP): Perplexity of a sentence is the measure of how well can it be predicted by the reader while reading. Low value of perplexity means the sentence is highly predictable and thus less perplexing or confusing to the reader. This is calculated using a language model.

We use the AWD-LSTM [MKS17] model for perplexity calculation. Though BERT([DCLT18]) and Turing-NLG([tnl20]) give better results for perplexity than AWD-LSTM, these two are complex models and have a lot of parameters. As a result of this, these take large amount of time and hardware resources to train and evaluate perplexity.

VReadA initializes every corpus (papers are arranged topic-wise in different cor-

pora) with an AWD-LSTM model pre-trained on English dataset from Wikipedia. This means that the model is well-versed with the nuances of English Grammar. The model is then fine-tuned by training it on the new papers that are archived into the corpus. This will make the model pick the corpus-specific details. The model can then be used to compute the perplexity.

(b) Coherence(COH): Coherence score of the sentence basically captures how well it is fitting in the context. High value of coherence means the sentence fits in the context nicely and is good for the understanding of the reader whereas a lower value implies that the sentence is not correctly fit at its position.

Inspired by the results of the Semantic Similarity Graph ([PT17]), we represent sentence as vectors(embeddings) and consider the semantic similarity while measuring coherence so as to take into account the related-yet-not-identical entities. We use pre-trained word2Vec [MSC+13] vectors for word embeddings and obtain the sentence embedding by averaging the vectors of the words present in the sentence. We propose two new ways of capturing coherence. These are:

- NGramSim: It is a directed graph with a parameter n. For every sentence, we establish an edge to the n sentences in front of it. That is, sentence i is connected to sentence j if j $\leq$ i+n. The weight of the edge between the sentence i and sentence j is given by:

$$w(s_i, s_j) = \frac{cos(\vec{s_i}, \vec{s_j})}{|i - j|} \tag{3.1}$$

where

cos($\vec{s_i}$,$\vec{s_j}$) is the cosine similarity between the 2 sentence vectors.

Fig. 3.1 shows a NGramSim graph.

- NWindowSim: It is similar to NGramSim but here we also connect the sentence to n sentences behind it. That is, sentence i is connected to sentence

j if i-n $\leq$ j $\leq$ i+n. The weight of the edge between the sentence i and sentence j is same as given by eq. 3.1. Fig. 3.2 shows a NWindowSim graph.

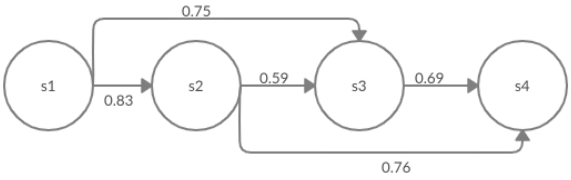

Figure 3.1: NGramSim graph for 4 sentences with n=2

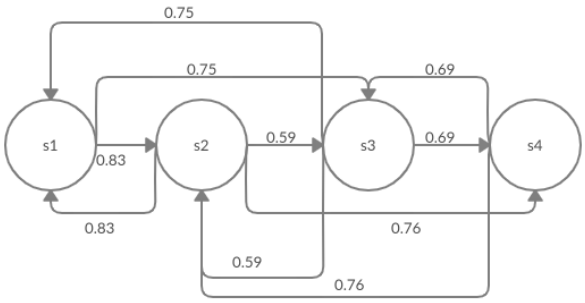

Figure 3.2: NWindowSim graph for 4 sentences with n=2

Once the graph is created, for any sentence, the coherence score is the average of the weights of the edges going out through it. The coherence score of the text can then found by taking the average of the coherence scores of its sentences.

To test the accuracy of the proposed methods, we tested them with the PAV, SSV and MSV methods as described in the Semantic Similarity Graph ([PT17]) approach. We took 2 articles A1 (on Cricket) and A2 (on Politics) and created a new Article A3 by combining A1 and A2 (concatenating A2 with A1). Then 100 different permutations were created for each of the three articles. The task is to differentiate the real article from the permutation. The coherence was calculated for all the permutations and compared with the real one. If the coherence score for a permutation is less than that for real, then it is a success as our model rightly concluded that the permutation is less coherent than the original article. All the methods were tried on all the 300 (permutation, article) pairs. The result for an article indicates how many of the 100

permutations of that article were given less (strictly less) scores than the real one (success count). Table 3.1 shows the result of the experiment.

It can be seen from the table 3.1 that our proposed methods are efficient in modelling coherence. NWindowSim performs way better than SSV and MSV and the best accuracy attained by NWindowSim (at n=2) beats the best accuracy attained by the methods mentioned in the Semantic Similarity Graph ([PT17]) approach.

		Article 1	Article 2	Article 3	Aggregate
SSV		77	64	98	79.67
	$\theta = 0.1, 0.2..0.7$	87	79	100	88.67
	$\theta = 0.8$	66	79	99	81.33
MSV	$\theta = 0.85$	79	8	99	62
	$\theta = 0.9$	90	78	100	89.34
	$\theta = 0.95$	90	78	100	79.67
	$\alpha = 0.1$	96	85	100	93.67
	$\alpha = 0.3$	95	87	100	**94**
PAV	$\alpha = 0.5$	92	88	100	93.34
	$\alpha = 0.7$	84	84	100	89.34
	$\alpha = 0.9$	71	86	100	85.67
	n = 1	96	82	100	92.67
NGramSim	n = 2	94	52	100	82
	n = 3	96	16	100	70.67
	n = 5	94	9	100	67.67
	n = 1	99	83	100	94
NWindowSim	n = 2	94	89	100	**94.34**
	n = 3	90	82	100	90.67
	n = 5	85	79	100	88

Table 3.1: Comparing the various coherence models

(The high accuracy obtained by the methods on article A3 can be attributed to the fact that it was a mix of two articles. And so, in any permutation of A3, there were sentences belonging to two different topics together which was easier for the models

to pick because of their higher dissimilarity.)

So, in light of the above experiments, we use NWindowSim with n = 2 to get the coherence score of the sentence. A sentence is connected with 2 sentences next to it and 2 sentences behind it. The weight for each such edge is given by the eq. 3.1. The coherence score of the sentence is then the average of these weights.

3.2 Storing and Analyzing Data

Assume that we have a corpus of nicely written scientific articles. These articles may be the ones which got viral, had many citations, were downloaded more and so on. The corpus provides us a benchmark for readability. We need to find a way to analyze a sample paper with respect to the corpus.

First, we process all the papers present in our corpus and calculate the above discussed 13 parameters for all the sentences present in the corpus. For each of the above parameters, we store the mean and standard deviation of the parameter's values for all the sentences in the corpus. Say we have N sentences in our corpus s_1,s_2 ... s_N and for each of these sentences, we calculate the average word length and let the average word lengths be w_1,w_2 ... w_N. So for the parameter Average Word Length (AWL), we store the mean of these average word lengths as μ_{AWL} and standard deviation of these average word lengths as σ_{AWL}. Similarly, the μ and σ values are stored for all the parameters.

Now, if say we have a text sample whose readability is to be analyzed. We do it one sentence at a time. Say we are at sentence 1 and analyzing the Average Word Length (AWL) and let AWL for this sentence be w. First, we will calculate the z-score of this value with respect to the distribution of AWL values in our corpus. This can be calculated as :

$$z = \frac{w - \mu_{AWL}}{\sigma_{AWL}} \tag{3.2}$$

where μ_{AWL} is the mean of AWL values of our corpus and σ_{AWL} is the standard deviation of the AWL values of our corpus.

The z-score basically tells us where our computed value w stands with respect to the AWL data. It measures the distance between w and the mean in steps of standard deviation. If z-score is 0, then it means that w is at mean, if it is 1 (or -1), it means that w is one standard deviation away from the mean. So if we get z-score for w as 0, w is equal to the average AWL of our corpus. Now as we are considering AWL and less AWL is good for readability, a z-score less than 0 would mean that the sentence has less AWL as compared to mean of the corpus and that's nice for readability whereas positive z-score would mean that the sentence has AWL more than the mean of the corpus and hence can be improved.

This though can not be generalized to say that lesser z-score is better for all the parameters. It is so because three of our parameters, namely Visual Words Ratio (VWR), Flesch Reading Ease (FRE) and Coherence (COH) indicate better readability with a higher value. That is, high values of VWR, FRE and COH would suggest that the sample has high readability. So for these 3 parameters, the more is the z-score, the better it will be for the sample sentence as it is performing better than the corpus (on average).

To incorporate this, we find the normalized z-scores (z'). Assume that on a sample sentence, we get the values s_{SL}, s_{PTH}, ... s_{COH} as the measures of the 13 parameters and using the respective means and standard deviations, we calculate z-scores for all these to be z_{SL}, z_{PTH}, ... z_{COH}. Now z' will be given by :

$$z_i\text{'} = \begin{cases} \text{-}z_i, & \text{if i is VWR, FRE or COH} \\ z_i, & \text{otherwise} \end{cases}$$

After normalizing the z scores to get z' scores, now we can generalize that less value of z' for a parameter is better for readability as it means that the sentence is performing better than the corpus (on average).

Let's say our samples had N sentences and for each of them, we find the 13 z' values to

get a matrix of size N X 13. Now, we will see how this matrix can be visualized.

3.3 Visualization

At the end of the last section, we got a matrix of size N X 13. Let us call this matrix M. The $(i, j)^{th}$ cell of M has the z' score for the j^{th} parameter computed on the i^{th} sentence. The more negative this value is, the better the i^{th} sentence performs on the j^{th} parameter.We visualize the matrix in 3 parts:

1. Sentence-wise z-score visualization: This a N X 13 size heatmap. We map every cell of M to a shade of the blue color. The greater the value, the darker will be the blue color and vice versa. So, if $(i, j)^{th}$ cell of the heatmap has dark blue color, it means its value is large and so the i^{th} sentence does not perform well on the j^{th} parameter. The columns of the heatmap are separated by the types of parameters, i.e. in 4 groups of Syntactic, Lexical, General and ML Based as discussed above.

   As greater values indicate poor readability, authors can work on sentences and sections which are darker and can improve them. This will enable them to increase the readability of their writing.

2. Distribution of Parameters: This is a box-and-whisker graph to plot the distribution of the normalized z-values for all the parameters. Box-and-whisker plot is a convenient way to display data distribution through quartiles. The author can see the range of the normalized z-values for all the parameters and can work on those parameters first whose range lies well ahead of 0 or the ones who have outliers having high positive value.

   The box plots of different types of parameters (Syntactic, Lexical, General and ML Based) are color coded differently. This can enable the author to concentrate on particular type of parameters first and gives a clearer representation of how the text fares on the different types of parameters.

3. Type-wise Summary: First, for all the thirteen parameters, the normalized z-values of all the sentences are averaged to get the average normalized z-value for all the parameters. Let these be n_{SL}, n_{PTH} .. n_{COH}. Then, for each of the types of parameters namely Syntactic, Lexical, General and ML Based, we compute the average of the average normalized z-values of all the parameters in that type. For eg., for Syntactic parameters, we will take the average of n_{SL} and n_{PTH} while for the ML Based parameters, we take the average of n_{PRP} and n_{COH}. This way, we will get 4 values, one for each of the parameter types. A heatmap is then plotted to visualize these 4 values. A darker color to the ML Based parameters cell will mean that the text sample does perform well on the ML Based parameters.

   This allows the author to have a high-level evaluation of the text on each of the parameter types. He/she can first focus on the parameter type whose cell is darker. Once a parameter type is fixed, the box-whisker plot and the sentence-wise plot can be used to see which parameters under the type have the most poor values and which sentences are responsible for them.

Fig. 3.3 shows the visualization a 14 sentences text sample produced as analyzed by VReadA. Overall the text is fine as indicated by the not-so-dark cells in the summary plot. The whisker plot has a positive outlier for Parse Tree Height(PTH) which can also be seen in the sentence-wise plot as sentence 12 in the PTH column. This means that sentence 12 is grammatically complex as it has a tall parse tree. This is validated by the fact that sentence 12 in the sample is "A recent addition to professional tennis has been the adoption of electronic review technology coupled with a point-challenge system, which allows a player to contest the line call of a point, a system known as Hawk-Eye.", which is indeed quite large and complex. Similar positive outlier is there in Part-Of-Speech(POS) as well and sentence 10 is the culprit as can be seen in the POS column in the sentence-wise graph. Similarly, SL and DWR have whisker plots extending deep in the positive region and sentence 14 and 13 can be observed as the ones that need to be amended (as they are

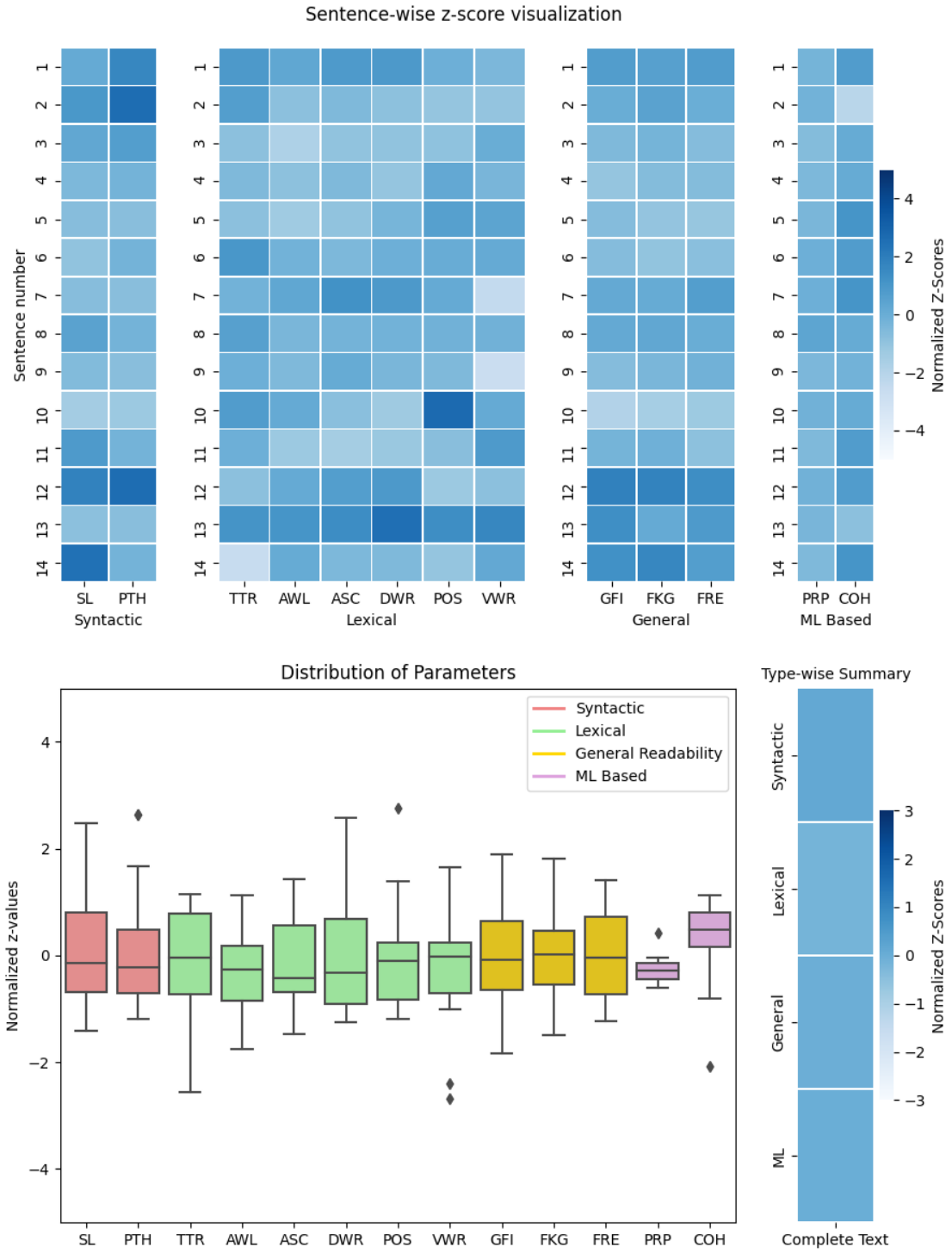

Figure 3.3: Visualizing readability for a sample with 14 sentences

darkest in the respective columns).

This shows that finding the areas with less readability is very easy with the help of VReadA. It can be used by authors while writing to get a feedback about their writing and improve the sentences marked poorly by VReadA, thus increasing the readability of their writing.

3.4 Results

We will now test VReadA's ability to analyze readability. A corpus is created which has 3 articles, one each on the topics Baseball, Tennis and Football. In each article, we have introductory paragraph of the Wikipedia page of that sport. Using this corpus, we perform the following three tests to check VReadA's ability:

3.4.1 General Analysis

First we do a general test. It will basically check VReadA's ability to distinguish between a less readable and more readable text sample. We take two sample articles. First one is the introductory paragraph of Cricket on its page on Wikipedia while second one is the introductory paragraph of Cricket on its page on Simple Wikipedia. Simple Wikipedia is a version of Wikipedia written in basic and easy English to be used by students, children and people who are new to English. This means that the second article will surely be more readable than the first article as it is on Simple Wikipedia and is meant to be simpler and easier to understand.

VReadA's output for the Wikipedia article is the fig. 3.4 and that for the Simple Wikipedia article is the fig. 3.5. The following can be inferred from the two visualizations:

- The topic-wise summary map is lighter for the Simple Wikipedia one as compared to that from the Wikipedia which is correct as we have seen that article from Simple Wikipedia is meant to be more readable and easier to understand.

- The box-whisker plots of parameters from the Simple Wikipedia article roughly lie from -1 to 1 while they lie from 0 to 2.5 for the Wikipedia article's visualization. This shows that the normalized z-values are higher for the Wikipedia article and so it is less readable, which is indeed correct.

- The sentence-wise visualization for the Wikipedia article is much darker than the Simple Wikipedia one (especially for the Syntactic, Lexical and the General Readability Parameters). This validates that Simple Wikipedia article is written with small and easy sentences using easy vocabulary for easier understanding and shows VReadA's power to judge readability accurately.

This test showed VReadA's ability to compare readabilities based on the syntactic and lexical parameters as the articles taken from the Wikipedia and Simple Wikipedia differed on these aspects only (Both were similar in coherence and perplexity). So, we test VReadA's ability to measure the coherence and perplexity in the next two tests.

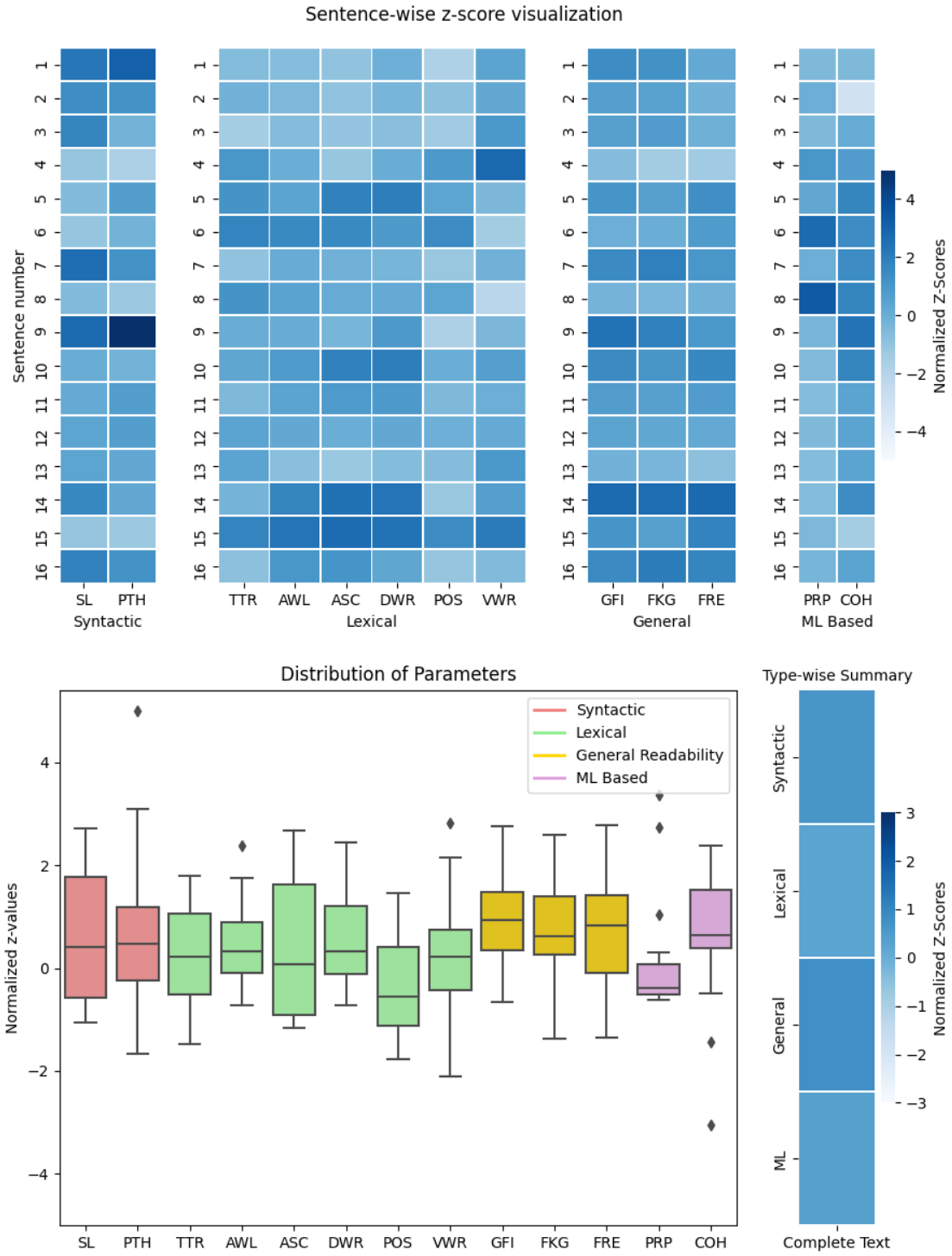

Figure 3.4: Visualization of the article taken from Wikipedia

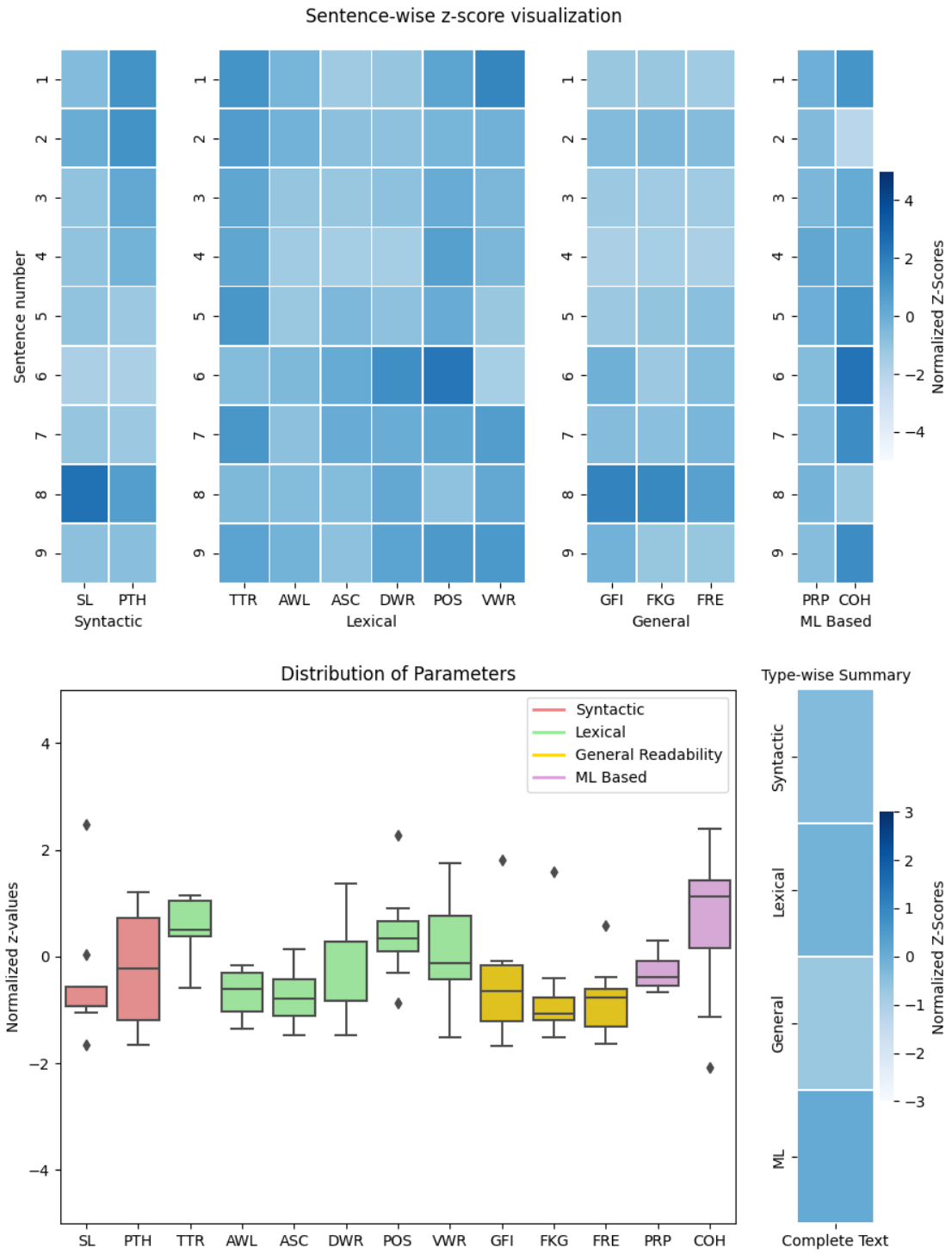

Figure 3.5: Visualization of the article taken from Simple Wikipedia

3.4.2 Perplexity Analysis

A sentence can be written with less words and using easy vocabulary but it is not necessary that it makes sense. For eg., consider the sentence "I games no play". This is a small sentence with easy vocabulary. So, it will be scored nicely on the syntactic and lexical parameters. But, it makes no sense. We can't predict any of it while reading. Here comes the role of perplexity. Such a sentence will have very high perplexity (it is perplexing for the reader as it is not understood easily) as the model will give a very less probability to it. This shows the importance of the perplexity parameter as it can help detecting such sentences. We will now test VRead's ability to judge such sentences.

We take the article on Cricket from Simple Wikipedia (used above). The fifth sentence in this article is "The area of play is a 30 yard circle inside the cricket ground or stadium.". We remove some of the connectives and reduce it to "The area of play 30 yard circle the cricket or stadium.". This way we do not worsen the sentence syntactically or lexically but we have made it confusing (or increased its perplexity). We are unable to understand what the new sentence wants to say. VReadA is then made to analyze the modified article. The obtained visualization is shown in the fig. 3.6. As compared to the original Simple Wikipedia's article, we see here that in the box-whisker plot for the perplexity(PRP) parameter, there is a outlier here (which was not there in the original visualization). Using the sentence-wise scores visualization, under the PRP column, we can see that VReadA indicates that the fifth sentence is the root of the problem (most dark in the column) which is indeed what we wanted.

This shows VRead is very effective is pointing out the sentences which may be scoring high on the syntactic and lexical parameters but do not portray any meaning or are "perplexing" for the reader to understand.

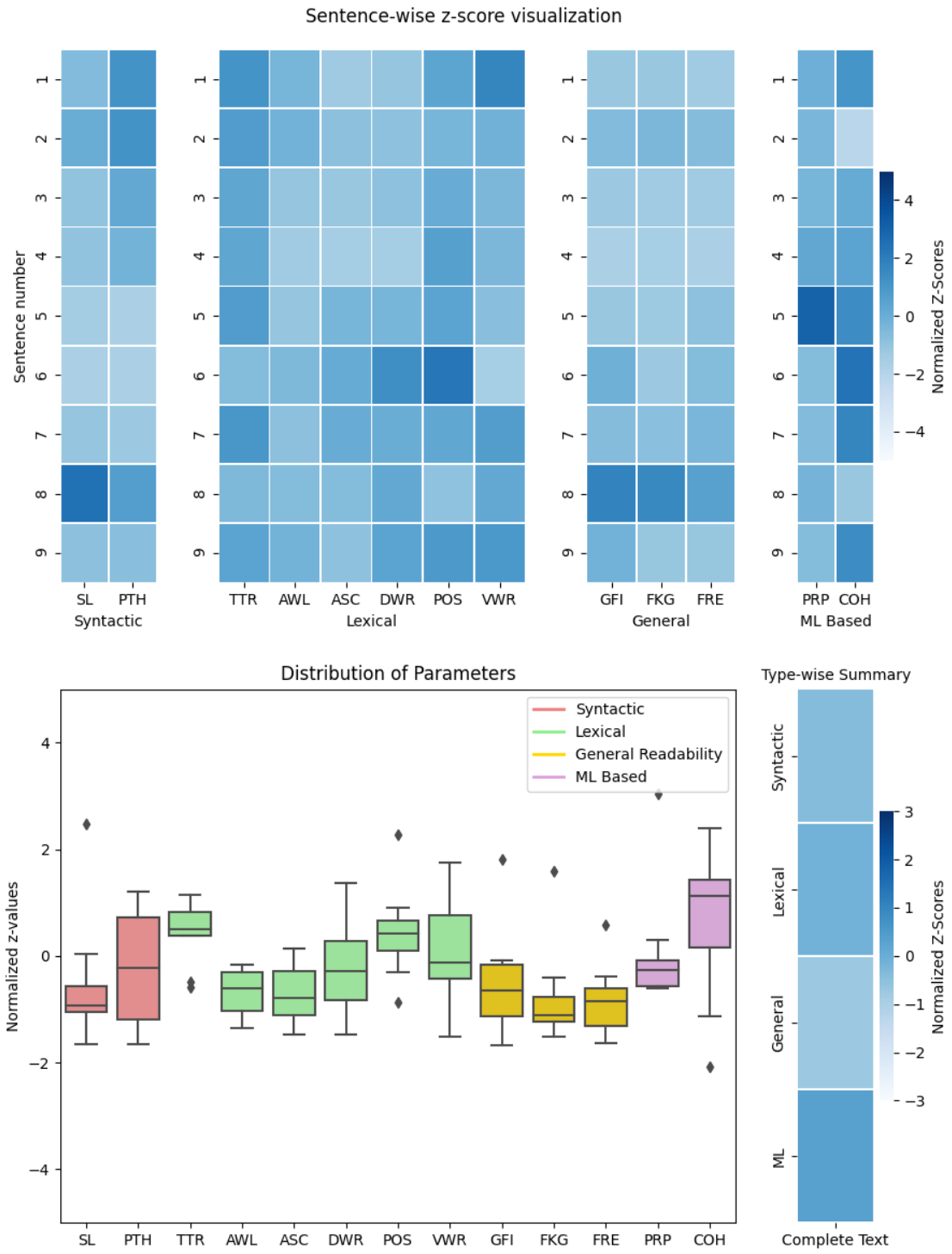

Figure 3.6: Visualization of the article taken from Simple Wikipedia after making the 5^{th} sentence perplex.

3.4.3 Coherence Analysis

This test will check VReadA's ability to judge the coherence of the sample. To test this, we take the article on Cricket from Simple Wikipedia (used above) and add the following two lines to it: "I love to eat oranges. The academic study of politics is referred to as political science.". These are added after the first sentence. As we can see, these two are off-topic and not related to cricket. Thus, this makes the article incoherent. These sentences are syntactically and lexically fine and are not perplex either as they make full sense. But, they are certainly wrong as they are not in the context of the article and this signifies the importance of the coherence parameter. Now let us see how VReadA deals with it. The article's visualization, as given by VReadA, is shown in the fig 3.7.

We observe that the box-whisker plot of the coherence parameter shows that something is wrong by extending all the way upto 6. In the sentence-wise normalized z-score visualization, it can be seen that the first three statements have very dark color. This indicates that they have very less coherence score and VReadA judges them as incoherent. This is indeed correct because they are of different topics (first statement is related to cricket, second is a general English statement and the third statement is related to politics) and so are truly incoherent. The rest of the statements are about cricket and so are coherent as can be judged by their light colors in the coherence column. This shows VReadA's effectivity in finding out the parts of a text sample which are incoherent.

The results of the three tests performed above show the effectivity of VReadA. It shows how VReadA does a rigorous and extensive analysis of a text sample covering many aspects like Syntactic and Lexical Complexity, Perplexity and Coherence. The visualization produced is very useful to find the areas (sentences) which need improvement. This ability of VReadA to detect and display the problematic areas accurately is what can be used by the authors to make their writings more readable using VReadA.

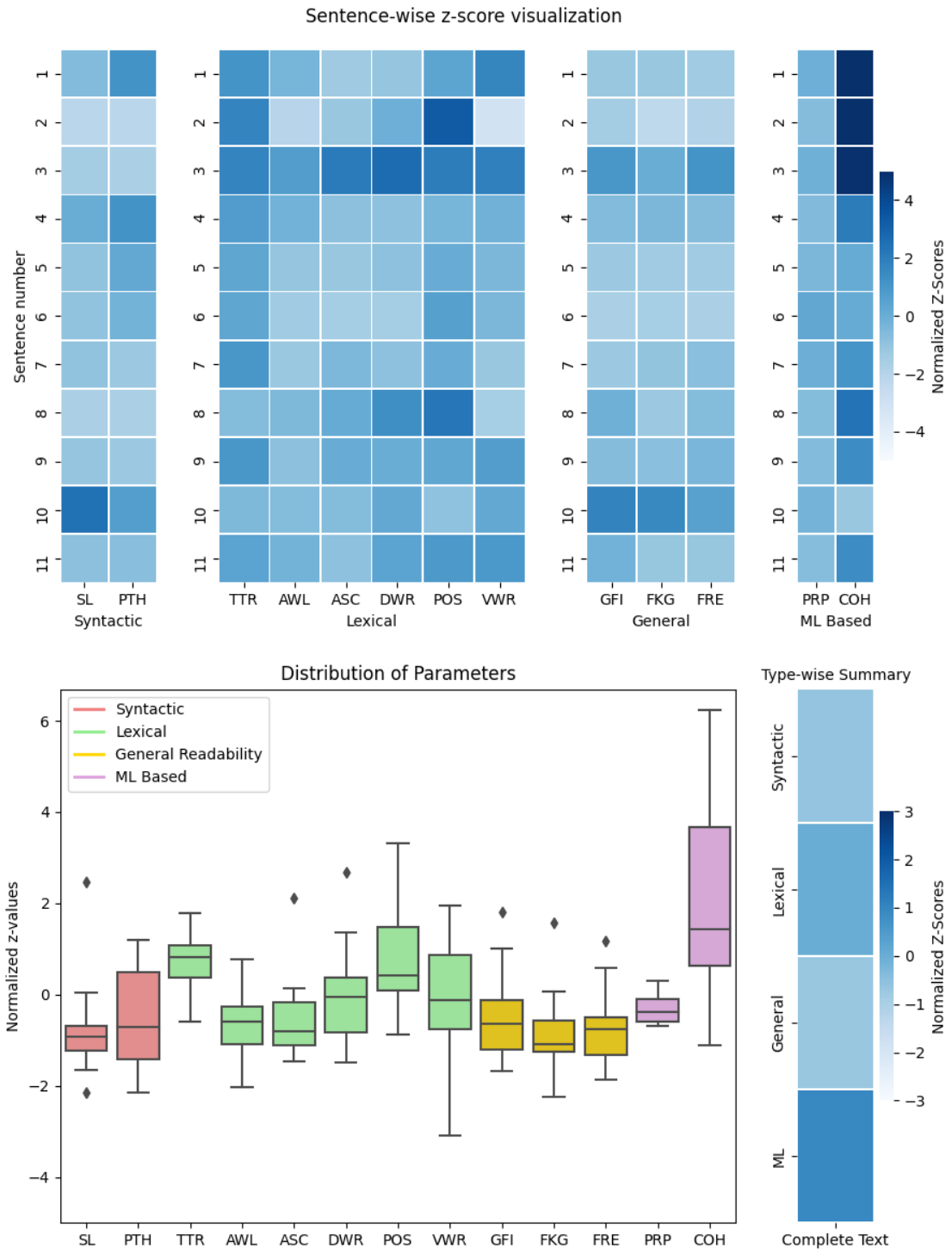

Figure 3.7: Visualization of the article taken from Simple Wikipedia after adding 2 off-topic sentences after the first sentence

Chapter 4

Conclusions and Future Work

We studied how readability impacted the success of scientific writing and did an extensive research on different measures of Linguistic complexity that affect readability which includes the Static Readability measures (accounting for the Syntactic and Lexical complexity) and the Machine Learning based measures (Perplexity and Coherence). The insights and ideas from the research were used to create VReadA, a tool that can be used for Visual Readability Analysis of text samples. VReadA analyzes the text sample rigorously on various aspects of linguistic complexity and readability and gives a detailed visualization of the analysis. This visualization enables the authors to easily detect the less readabble parts of the text which are graded poorly by VReadA. This feedback can be used by the authors to make their writings more readable thus increasing the chances of the paper/article being selected for a journal and getting cited more. The results showed that VReadA is indeed quite effective in analyzing the readability and pointing out the less readable parts.

There are some things which can be tried and used to improve VReadA's analysis. These are :

1. For Coherence Modelling, we are using pre-trained word2vec ([MSC+13]) embeddings for the words and averaging them to get the sentence embedding. We could try using doc2vec ([LM14]) embeddings trained on our corpus. This might help the coherence

model learn the semantics of a sentence more accurately which will in turn improve its analysis of coherence.

2. For Perplexity Analysis, we are using the AWD-LSTM ([MKS17]) model which is pre-trained on English dataset and we are fine-tuning it using the papers we have in our corpus. We can try using the BERT ([DCLT18]) language model. For this, we could use Allen Institute for AI's SciBERT ([Sci]), which is a BERT model trained on a large corpus containing scientific texts. This could improve VReadA's perplexity analysis.

3. VReadA has the option of analyzing the document sentence wise only. Option for paragraph level and corpus level analysis can be added. This can be used by authors to first find the paragraph which is the most problematic (scored poorly by VReadA) and then the sentences of this paragraph can be visualized individually to find the less readable sentences which can be corrected in the subsequent drafts.

References

[BGJM16] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[BL04] Regina Barzilay and Lillian Lee. Catching the drift: Probabilistic content models, with applications to generation and summarization. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 113–120, Boston, Massachusetts, USA, May 2 - May 7 2004. Association for Computational Linguistics.

[BL08] Regina Barzilay and Mirella Lapata. Modeling local coherence: An entity-based approach. *Computational Linguistics*, 34(1):1–34, 2008.

[CLZZ17] Baiyun Cui, Yingming Li, Yaqing Zhang, and Zhongfei Zhang. Text coherence analysis based on deep neural network. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, page 2027–2030, New York, NY, USA, 2017. Association for Computing Machinery.

[Col13] Michael Collins. Language Modelling, 2013. URL: `http://www.cs.columbia.edu/~mcollins/lm-spring2013.pdf`. Last visited on 2019/11/08.

[CTC04] Kevyn Collins-Thompson and James Callan. A language modeling approach to predicting reading difficulty. pages 193–200, 01 2004.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[EAC07] Micha Elsner, Joseph Austerweil, and Eugene Charniak. A unified local and global model for discourse coherence. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*, pages 436–443, Rochester, New York, April 2007. Association for Computational Linguistics.

[FLH14] Vanessa Wei Feng, Ziheng Lin, and Graeme Hirst. The impact of deep hierarchical discourse structures in the evaluation of text coherence. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 940–949, Dublin, Ireland, August 2014. Dublin City University and Association for Computational Linguistics.

[GPL12] Marco Guerini, Alberto Pepe, and Bruno Lepri. Do linguistic style and readability of scientific abstracts affect their virality? 03 2012.

[GS13] Camille Guinaudeau and Michael Strube. Graph-based local coherence modeling. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 93–103, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

[HSK+12] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[JM00] Daniel Jurafsky and James Martin. *Speech and Language Processing*. 01 2000.

[JVS+16] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. 02 2016.

[LB05] Mirella Lapata and Regina Barzilay. Automatic evaluation of text coherence: Models and representations. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, page 1085–1090, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[LBD+19] Chao Lu, Yi Bu, Xianlei Dong, Jie Wang, Ying Ding, Vincent Larivière, Cassidy Sugimoto, Logan Paul, and Chengzhi Zhang. Analyzing linguistic complexity and scientific impact, 07 2019.

[LM14] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[LN13] Annie Louis and Ani Nenkova. What makes writing great? first experiments on article quality prediction in the science journalism domain. *TACL*, 1:341–352, 12 2013.

[MKS17] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. *CoRR*, abs/1708.02182, 2017.

[mrc] MRC Psycholinguistic Database. URL: `http://websites.psychology.uwa.edu.au/school/MRCDatabase/uwa_mrc.htm`. Last visited on 2019/11/08.

[MSC+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, G.s Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26, 10 2013.

[Ola15a] Christopher Olah. Understanding LSTM Networks, 2015. URL: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-rolled.png`. Last visited on 2019/11/08.

[Ola15b] Christopher Olah. Understanding LSTM Networks, 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-unrolled.png. Last visited on 2019/11/08.

[Ola15c] Christopher Olah. Understanding LSTM Networks, 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/img/RNN-longtermdependencies.png. Last visited on 2019/11/08.

[Ola15d] Christopher Olah. Understanding LSTM Networks, 2015. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-chain.png. Last visited on 2019/11/08.

[OSSK10] Daniela Oelke, David Spretke, Andreas Stoffel, and Daniel Keim. Visual readability analysis: How to make your writings easier to read. pages 123 – 130, 11 2010.

[PSM14] Jeffrey Pennington, Richard Socher, and Christoper Manning. Glove: Global vectors for word representation. volume 14, pages 1532–1543, 01 2014.

[PT17] Jan Wira Gotama Putra and Takenobu Tokunaga. Evaluating text coherence based on semantic similarity graph. In *Proceedings of TextGraphs-11: the Workshop on Graph-based Methods for Natural Language Processing*, pages 76–85, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[Sci] A BERT model for scientific text. URL: https://github.com/allenai/scibert. Last visited on 2020/05/12.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[tnl20] Turing-NLG Model by Mircosoft, 2020. URL: https://www.microsoft.com/en-us/research/blog/turing-nlg-a-17-billion-parameter-language-model-by-microsoft/. Last visited on 2020/05/10.

[w2v] Introduction to Word Embedding and Word2Vec. URL: https://bit.ly/35TY4kc. Last visited on 2020/05/12.

[WZZ+13] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.