The effect of noise in the training of convolutional neural networks for text summarisation

Master’s Thesis

Ailsa Meechan-Maddon

Uppsala University
Department of Linguistics and Philology
Master Programme in Language Technology
Master’s Thesis in Language Technology, 30 ECTS credits
June 5, 2019

Supervisors:
Christian Hardmeier, Uppsala University
Håkan Bårman, Inovia AI
Abstract

In this thesis, we work towards bridging the gap between two distinct areas: noisy text handling and text summarisation. The overall goal of the paper is to examine the effects of noise in the training of convolutional neural networks for text summarisation, with a view to understanding how to effectively create a noise-robust text-summarisation system. We look specifically at the problem of abstractive text summarisation of noisy data in the context of summarising error-containing documents from automatic speech recognition (ASR) output. We experiment with adding varying levels of noise (errors) to the 4 million-article Gigaword corpus and training an encoder-decoder CNN on it with the aim of producing a noise-robust text summarisation system. A total of six text summarisation models are trained, each with a different level of noise. We discover that the models with a high level of noise are indeed able to aptly summarise noisy data into clean summaries, despite a tendency for all models to overfit to the level of noise on which they were trained. Directions are given for future steps in order to create an even more noise-robust and flexible text summarisation system.
Contents

1 Introduction 4

2 Related Work 6
  2.1 Noisy text summarisation 6
  2.2 Text summarisation and neural networks 7
  2.3 Errors in ASR 8
  2.4 Artificial error generation 9

3 Data creation 11
  3.1 Gigaword Corpus 11
  3.2 Data augmentation and the creation of noise 11
    3.2.1 Pre-processing 13
    3.2.2 Text-to-Speech 13
    3.2.3 Speech recognition 14
    3.2.4 Noise analysis 14
    3.2.5 Error creation and addition 16

4 Text summarisation system 22
  4.1 Fairseq: a Seq2Seq approach to text summarisation 22
  4.2 Hyperparameters 23
    4.2.1 Optimiser 24
    4.2.2 Learning Rate 25
    4.2.3 Final hyperparameters 26
    4.2.4 Training environment 27

5 Environment 28
  5.1 Evaluation method 28
  5.2 Goals and hypotheses 30

6 Results 32

7 Discussion 34
  7.1 Analysis of results 34
    7.1.1 Outcome of hypotheses 34
    7.1.2 Individual models 35
    7.1.3 Overall 36
    7.1.4 Result examples 38
  7.2 The problem with text summarisation evaluation 41

8 Conclusion 44
Text summarisation is the process of creating a shorter text representing the key information from one or more input documents. This is usually carried out as a means of providing readers with the information they seek or wish to know in the most efficient and effective manner. In the present day, with such a vast amount of information readily available on the Internet, we find ourselves with an increasing need for such information to be conveyed in the most efficient and time-saving fashion. Manual summarisation is not an option due to the vast magnitude of data and the work involved; it “requires a considerable number of qualified unbiased experts, considerable time and budget and the application of the automatic techniques is inevitable with the increase of digital data available world-wide” (Yousefi-Azar and Hamey, 2017). The need for text summarisation is thus an extremely pertinent and prominent issue in the field of natural language processing (NLP), with much work currently being focused on this area (Almazaydeh, 2018; Patel and Chhinkaniwala, 2018; Wu et al., 2019; Yousefi-Azar and Hamey, 2017).

Thus far, the majority of research and work on text summarisation has been based on clean data, that is, documents with few spelling or grammatical errors. As fields such as automatic speech recognition (ASR) and text summarisation begin to merge, however, this becomes a problem; ASR systems are notoriously prone to transcription errors due to, among other reasons, speaker variation, poor recording environment and low technology levels within the ASR system itself. Therefore, there is an increasing need to produce a noise-robust text summarisation system.

Generally speaking, text summarisation systems are classified by three main properties: input, output and purpose. For input, the aim is either to summarise a single document or multiple documents into short summaries. For output, the goal is to produce either an extractive or abstractive summary. The extractive approach typically focuses on directly cutting out specific sentences from the input text and putting them back together to form a summary, while the abstractive method uses more complex methods to reword and reformulate the input into a more coherent summary, usually with the use of unseen — in relation to the input document vocabulary — words. The purpose of text summarisation can vary from being focused on a specific domain, to summarising a document to fit a specific query and finally to generic summarisation. These three classifications are exemplified in Figure 1.1.

As far as this paper goes, we select the follow classifications:

- **Input**: Single document. The paper will focus on the summarisation of single short articles.
- **Output**: Abstractive. The aim will be to create complex yet coherent summaries.
• **Purpose**: Generic. The application of this paper will deal with general-purpose text summarisation.

In this paper, we first carry out a small pilot study to determine the types of errors that ASR systems are most prone to making when transcribing speech. Based on the results of this, we create a statistical model to artificially generate and spread similar errors throughout the large, 4 million-article Gigaword corpus being used in this paper. Five separate datasets are created this way, each with varying amounts of errors, along with one totally clean dataset created from the original corpus. Each dataset consists of a training, validation and test set. The training set of each dataset is used to train an encoder-decoder CNN and then each of the created models is tested on all six of the available test sets.

We carry out the work in this paper in the context of a commercial environment. We work with Inovia AI\(^1\), a Swedish company, focusing on artificial intelligence solutions, who works with, among other things, ASR. They have a need for a summarisation system that is robust against the noise of ASR output and so the work in this paper will contribute to solving this task. From a broader perspective, such noise-robust text summarisation systems are becoming increasingly valuable in commercial environments as the accessibility of ASR systems increases.

The remainder of this paper is structured as follows: chapter 2 provides an overview of related work, specifically in the domains of noisy text summarisation (2.1), text summarisation and neural networks (2.2), errors in ASR (2.3) and artificial error generation (2.4). Chapter 3 covers the paper’s experimental setup as well as the processes undertaken in order to prepare the required data. Chapter 4 presents the sequence modelling toolkit used to create text summarisation models in this paper as well as details of the hyperparameters used. Chapter 5 gives an overview of the experimental environment and chapter 6 presents the results of the main experiments. Chapter 7 discusses the results in detail and their implications, while chapter 8 concludes the paper while pointing to areas of future research.

---

\(^1\)https://inoviagroup.se/
2 Related Work

2.1 Noisy text summarisation

Within the field of text summarisation, “the focus to date has typically been on clean, well-formatted documents, i.e., documents that contain relatively few spelling and grammatical errors such as news articles or published technical material” (Jing et al., 2003). That is, the effects of error-filled (noisy) text on the effectiveness of text summarisation is an area that has been relatively sparsely studied or examined, with Jing et al. (2003) being the only published paper to focus specifically on noisy text summarisation. Their paper studies the effects of using noisy text from documents scanned by optical character recognition (OCR) software. During this study, they carry out a pilot study by creating noisy versions of documents by using, for instance, excessively dark or excessively light photocopies. Furthermore, they add extra noise to certain documents by making single character substitutions (deletions, insertions and substitutions) at varying rates. They discover that “[t]he quality of summarisation is directly tied to the level of noise in a document”. They stress the importance of handling noise since “the output from OCR and speech recognition (ASR) systems typically contain various degrees of errors, and even purely electronic media, such as email, are not error-free”. While Jing et al. (2003) focus primarily on noise from OCR data, one of the most fundamental points of the paper is also true of ASR data; the data is run through such an extensive pipeline that there is no shortage of potential sources of errors. Since text summarisation is usually the final stage in such a pipeline, the amount of noise in the final output accumulates throughout the previous steps; this accumulation accentuates the need for a robust text summarisation system. They suggest that one way to combat the challenges of noisy document summarisation would be to “retrain an existing system on noisy documents so that it will be more tolerant of noise”; this is precisely the aim of this paper.

Other fields within NLP have also been working on parsing and understanding noisy documents. Miller et al. (2000) used named-entity extraction as a means to gather key information from ASR-transcribed documents. While they garnered impressive results in terms of name finding in noisy data by utilising hidden Markov models, they stress that “[p]erformance on both speech and OCR input degrades linearly as a function of word error”, meaning that naturally, as WER increases, information extraction accuracy decreases. Such NLP techniques as named-entity recognition are often used as building blocks to reinforce noisy text summarisation systems. The general idea is to, as a preliminary step, extract named-entities from a noisy document. In essence, by discarding as much of the noise as possible and retaining only the key information, in a second step, a summary may be reconstructed around this in order to form full summaries. Roy and Subramaniam (2006) did exactly this; they carried out a study in which call centre phone conversations were transcribed using an ASR system. Due to the sound
quality of the recordings, the transcripts incurred a high word error rate (WER) of 40%. Despite the significant WER, the authors managed to produce useful summaries of the calls by exploiting automatic taxonomy generation (ATG), a method that relies “on parsing [...] to extract relationships between key concepts within the domain” and then linking them together to form an ontology. This approach is specific to domain-specific applications of text summarisation since it allows for inter-document redundancy; the idea that “[o]ne can give relevance to an idea depending on how redundant it is across documents” (Carrillo-Mendoza et al., 2017). The principle behind this is that the subjects of calls routed to any one call centre will naturally form a finite group. For instance, in a mobile phone carrier’s call centre, customers contacting the call centre will only be concerned with a select number of topics which could include, for example, upgrading their phone plan, problems with billing or a request for technical support. What these calls would not include, on the other hand, would be an inquiry into which type of wallpaper would work best in their bedroom. By working from this assumption and specifically seeking out keywords that are known to be particularly common across calls, the next stage can be carried out: an inverse redundancy system is employed to extract the most common and pertinent information and discard the rest. While this strategy is, of course, only effective in text summarisation systems in which there is a finite number of possible topics (as opposed to a general purpose text summarisation system), it has been shown to be particularly effective in constructing summaries (Roy and Subramaniam, 2006).

While their aim was not to summarise, Chalamalla et al. (2008) also work on a similar approach to extract important discourse patterns from noisy call centre ASR data. They first employ a rule-based method to extract questions and then cluster the questions based on commonly-occurring features using such techniques as patterning mining and named-entity recognition. This allows for conversations relating to similar subjects to be pooled together. Such techniques allow for the main topic to be identified, thus leaving room for an ontology to be built around it.

Hori and Furui (2001) study specifically the summarisation of noisy documents by working from a scoring system that scores potential summary candidates based on several criteria including significance scoring, linguistic likelihood, word confidence measures and word concatenation probability. Similar to Chalamalla et al. (2008) and Roy and Subramaniam (2006), the focus is on removing redundant information (either noise or invaluable words) and extracting the keywords and information instead, and then forming a coherent summary from this. While these approaches to text summarisation perform well under the test conditions described, given that they are rule-based, their results tend to rely heavily on domain-specific information, and the approaches involved are often somewhat constraining and static. Thus, we look to more dynamic and powerful methods.

### 2.2 Text summarisation and neural networks

More recent approaches to text summarisation typically take a neural network-centred approach (Nallapati et al., 2016; Yousefi-Azar and Hamey, 2017). Currently, one of the most common neural network (NN) approach to text summa-
sation is an encoder-decoder recurrent neural network (RNN). Such RNNs are commonly used in machine translation (MT) and consist of two separate RNNs; “[o]ne RNN encodes a sequence of symbols into a fixed-length vector representation, and the other decodes the representation into another sequence of symbols.” (Cho et al., 2014). In text summarisation, the encoding refers to the input of the document to be summarised; this is then decoded into the output — the summary. Nallapati et al. (2016) used such an encoder-decoder RNN to carry out abstractive text summarisation on several datasets including the Gigaword corpus (Graff et al., 2003) and achieved promising results by adding a state-of-the-art switching generator-pointer system to handle OOV words as well as an attention mechanism.

It must be noted, however, at the time of writing, little work has been done to mitigate the effects of noisy data in conjunction with neural network-based text summarisation. This paper aims to consolidate these areas and investigate the effects of noisy data on text summarisation.

2.3 Errors in ASR

ASR systems are notorious for being susceptible to errors related to — but not limited to — speaker variation and sound recording quality. As Errattahi et al. (2018) observe, ASR performance factors can be grouped into the following three categories:

• **Speaker variability:** This is related to the acoustic variation that commonly occurs within speakers due to emotion, tiredness, age and other factors. The acoustic model trained to capture speakers’ voices “is obtained using a limited amount of speech data that characterizes the speakers at a given time and situation” and thus usually does not capture this variability (Errattahi et al., 2018).

• **Spoken language variability:** The range of speakers using a single ASR system can include a variety of dialects and accents (due to different first languages); acoustic models are often trained on a finite set of speakers that do not always capture this level of variation.

• **Mismatch factors:** A significant issue in ASR is the disparity between recording conditions at the time of acoustic model training and those being used at test time. This can result from background noise, speech signal variation and transmission channels, among others.

ASR systems rely on both an acoustic model and a language model; the acoustic model is what interprets the speech signals while the language model works to ensure a grammatical and coherent output. The errors caused by the factors outlined above vary from system to system. Acoustic models are generally responsible for producing an accurate estimation and interpretation of the speech signals. In some cases, without a strong language model ensuring that the system’s output is both grammatical and spell-checked, errors can occur at a phonetic level and produce incorrectly-spelled words. Extremely strong language models, on the other hand, can also produce extremely confusing outputs; while trying to compensate for
an incorrectly-interpreted word, they may change a whole sequence of words. Thus, both a strong acoustic and language model are required to produce accurate output.

### 2.4 Artificial error generation

The field of artificial error generation (AEG) in NLP revolves primarily around its applications in foreign-language learning, although it is becoming increasingly present in other NLP areas such as text summarisation and ASR. Error-filled data is a pre-requisite for many applications related to error detection. However, there is currently little readily available data or resources and, as Rei et al. (2017) notes, “[s]hortage of available training data is holding back progress” in this area. One of the main issues is that the training data errors needed for one application are quite likely to be different from those needed for another application; that is, the type of errors required for a task related to detecting errors in foreign speakers’ writing will be quite distinct compared to those needed for evaluating ASR errors. Thus, when creating or seeking out error-filled data, one must ensure that such data fits the end purpose; as Rei et al. (2017) correctly observe, “naive methods for error generation can create data that does not resemble natural errors, thereby making downstream systems learn misleading or uninformative patterns.”

Generally, approaches to AEG can be split into one of two techniques: “[e]rrors can be injected into candidate texts using a deterministic approach (e.g. fixed rules) or probabilities derived from manually annotated samples in order to mimic real data” (Felice and Yuan, 2014). As will be seen in chapter 3, this paper will use a blend of these two techniques, using observations from manually annotated samples to create both probabilities and fixed rules.

Sjöbergh and Knutsson (2005) take a rule-based approach to generate errors as a means to creating an error detection system. This was based primarily on splitting compound words (which are common in Swedish, the focus language of the paper) and by creating agreement errors (for example “I bought a car” -> “I bought a cars”). While no formal evaluation of the AEG part of the paper was carried out, the authors comment that this approach worked very well, with only a small number of observed false negatives.

Rei et al. (2017) use two distinct AEG approaches in an attempt to create training data for error detection. The first is pattern extraction and focuses on identifying patterns and probabilities of incorrect sequences from annotated data and then applying those observations to a larger corpus. Their second approach is to treat the problem as a machine translation task; that is, to train a statistical machine translation (SMT) system on correct-incorrect sentence pairs. They discovered that a combination of both of these techniques performed best by exploiting the fact that the two systems complemented one another; this approach beat out previous state-of-the-art results.

Most work done on AEG seems to focus on creating error-filled datasets in order to correct grammatical mistakes made by learners of English (see Felice and Yuan (2014)) and there appears to be no work that concentrates specifically on AEG in relation to ASR errors. Thus, the errors used as the base of this work will be primarily based on a set of initial experiments...
carried out (see section 3.2), designed to create a sample of ASR errors and also on knowledge of typical ASR errors (see section 2.3).
3 Data creation

There currently exists no noisy and readily available training data for text summarisation tasks. As section 2.4 recalls, one of the main issues is drawn from the fact that the type of noise required for one error-oriented application tends to differ quite vastly from that needed for another. In this sense, we must obtain data that imitates — as accurately as possible — the type of noise one can expect from ASR output. Thus, it was an important preliminary task to create the relevant data for the paper’s purposes. The steps and tools used for this are outlined in this section.

3.1 Gigaword Corpus

Throughout the experiments in this paper, the Gigaword corpus (Graff et al., 2003; Napoles et al., 2012) is used. The Gigaword corpus has become an industry standard for training and testing text summarisation systems. We use a version that was adapted by Rush et al. (2015) specifically for text summarisation system development. It consists of a total of 4 million short news ‘articles’ each with a corresponding title. The articles consist of the first sentence of the full news article, while the titles are their headlines and are around 4-7 words in length. The titles are considered to be summaries of the articles. In order to train and test text summarisation models in this paper, the Gigaword corpus was split into training, validation and test sets consisting of 3.6M, 180K and 190K words, respectively. This is similar to splits used in Nallapati et al. (2016) and Y. Zhang et al. (2019).

3.2 Data augmentation and the creation of noise

In order to train text summarisation models that were robust to noise, it was first necessary to create a noisy dataset. The type of noise needed was extremely crucial, as it had to accurately reflect the real-life noise found in ASR transcriptions. The methods used to determine such errors and recreate them on a large scale are outlined in this section.

While using a text-to-speech (TTS) and ASR pipeline to transform the entire Gigaword corpus would have been a more optimal solution, given the 4 million article size of the corpus, the time needed to carry out these tasks would have been out of the time frame of this paper. Thus, we determined instead two methods to simulate the errors formed in ASR. We included two methods of data collection at this stage would maximise both the quality and quantity of error generation data in the most efficient way. The two methods can be described as follows:

- **Method 1:** A small random sample (200 articles) from the Gigaword corpus was extracted. Voice recordings of the articles were synthesised by using
TTS technology and synthetic noise was added to these recordings. ASR was then used to transcribe the noisy recordings. The pipeline by which we generate artificial ASR samples of the Gigaword corpus in this way is outlined in Figure 3.1.

- **Method 2:** Similar to the first method, a small random sample of fifty articles was taken from the Gigaword corpus. These fifty articles were then manually recorded by a human. Following this, the fifty recordings were transcribed by using ASR with no noise addition. The results of these are used as a baseline of ASR errors.

The resulting transcripts from these two methods were then manually compared to the original dataset and the most common errors in voice recognition were noted to be used and applied to the entire Gigaword dataset in later steps; this process is described in more detail in section 3.2.4. Of course, it must be emphasised that the end results here — the output transcripts — are simply simulations of real-life speech-to-text transcripts. Due to the large number of samples required, an automated pipeline was considered to be the most effective option. However, we took several precautions to ensure a realistic simulation, from state-of-the-art speech synthesis to the addition of noise to the recordings to imitate genuine speech samples.

**Figure 3.1:** The pipeline by a voice recognition environment was simulated on 200 randomly selected samples from the Gigaword corpus.
3.2.1 Pre-processing

The available version of the Gigaword corpus\(^1\) (Rush et al., 2015) has had several normalisation techniques applied to it, all of which conform to the Penn Treebank 3 (Marcus et al., 1999) style of tokenisation. For instance, all punctuation has been tokenised and is thus separate from words. Furthermore, some delexicalisation has been carried out: tokens that appear fewer than 5 times have been replaced with an \(<\text{UNK}>\) token. In addition to this, all numbers have been replaced with a pound (\(#\)) sign. While these modifications are useful in making the corpus more accessible to other NLP operations, they hinder the pipeline in Figure 3.1 as, for instance, the punctuation causes unnecessary pauses and unnatural intonation during the text-to-speech process, and the replaced tokens produce odd language model outputs at the voice-to-text stage.

Thus, before proceeding, the 250 randomly-selected sentences (200 from method 1 and 50 from method 2) were pre-processed in order to ensure that they resembled standard sentences in the most natural way and to optimise them for speech synthesis/recording. A lot of this pre-processing was carried out by using Stanford CoreNLP tools (Manning et al., 2014) which feature a parameter for effectively undoing such tokenisation (‘-untok’). The remainder was carried out with a normalisation code that was specific to the unwanted tokens.

3.2.2 Text-to-Speech

**Method 1:** For the TTS portion of the pipeline described in Figure 3.1, Google Cloud’s Text-to-Speech\(^2\) was used. Google is a technology company that includes a wealth of different technologies from search engines to ASR and TTS services. They acquired the artificial intelligence company Deepmind in 2014. Google Cloud’s Text-to-Speech system uses Deepmind’s WaveNet (Oord et al., 2016) technology to create what is widely considered as one of the most state-of-the-art and human-like TTS systems available (Gu and Kang, 2018; Paine et al., 2016; J. Zhang et al., 2018). Each of the randomly selected and pre-processed 200 articles from the Gigaword training set was sent to Google’s Text-to-Speech API and sent back as an MP3 file. To simulate a more realistic environment, half of the 200 sentences were set to return a female voice (voicename: \textit{en-US-Wavenet-C}) while the other half returned a male voice (voicename: \textit{en-US-Wavenet-A}). For both, the American English voice setting was utilised (language code: \textit{en-US}). This process produced MP3 files which were then converted to WAV files using SoX\(^3\).

**Method 2:** The 50 randomly-selected sentences in method 2 were recorded by the author and thus Google Cloud’s Text-to-Speech was not utilised. The author (a native English speaker with a mild Scottish accent) used the built-in microphone of an Apple MacBook Pro (2016 edition) and Audacity\(^4\) software for the recording process. There was no environmental noise. This process produced WAV files as output.

---

\(^1\)https://github.com/harvardnlp/sent-summary  
\(^2\)https://cloud.google.com/text-to-speech/  
\(^3\)https://github.com/chirlu/sox  
\(^4\)https://www.audacityteam.org/
3.2.3 Speech recognition

For the voice-to-text part of the pipeline, the WAV files created in the previous stage were used as input.

**Method 1:** Since Google’s Text-to-Speech system produces extremely clear voice samples, it was necessary to add noise to them in order to simulate a more realistic recording environment as well as a less advanced speech recognition system. Some preliminary experiments where carried out using SoX software to add varying levels of both white and pink noise. The aim was to distort the WAV file somewhat while still ensuring that it was, to some degree, comprehensible. In the end, pink noise at a volume 70% (based on SoX’s metrics) as we found that it distorted the speech without making it incomprehensible, and all in all produced output similar to that of ASR output. Pink noise, which has equal power per octave, was used to imitate the effects of microphone static in order to realistically recreate the reduction in audio quality due to recording on a microphone. As part of the pipeline, this setting of pink noise was added to each of the 200 WAV files. Without adding noise, few — if not zero — errors would occur.

**Method 2:** No noise was added to the 50 human-recorded sentences; since the recordings were totally organic, they included standard microphone recording static already and were not formed from synthesised speech, the addition of further noise was not required in order to generate standard ASR errors.

**Both methods:** The final WAV files from each method were then sent to Google Cloud’s Speech-to-Text, a state-of-the-art speech recognition system using a sample rate of 24000Hz, language code ‘en-US’, and a linear pulse-code modulation encoding was used (encoding: LINEAR16). The best transcripts are extracted from Google’s output and written into a file in the same order and format as the original randomly selected Gigaword articles in order to facilitate comparison.

3.2.4 Noise analysis

At this stage, for each of the two methods outlined in section 3.2.3, two versions of each set of randomly-selected sentences from the Gigaword corpus were available: the original set and the ‘noisy’ set. The noisy set was manually compared to the original set in order to gauge the types of errors that may occur during speech recognition.

A tally was kept with regards to which errors occurred. Only errors that occurred more than 5 times within the 250 analysis articles were accepted to be reproduced across the entire corpus.

It must be noted that the 100 sentences recorded with a female voice were more susceptible to errors and misinterpretations than the 100 male voice recordings due to the fact that female voices have less strength at lower frequencies and thus are more likely to be masked by the pink noise. On average, there were ∼ 20% more errors in female speech compared to male speech.

---

5https://github.com/chirlu/sox
6https://cloud.google.com/speech-to-text/
Table 3.1: Errors noted during the noise analysis of section 3.2.4

<table>
<thead>
<tr>
<th>Errors</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>'s ↔ Ø</td>
<td>Genitive case was added/removed</td>
</tr>
<tr>
<td>Plural ↔ non-plural</td>
<td>Switching of plurality</td>
</tr>
<tr>
<td>its ↔ it</td>
<td></td>
</tr>
<tr>
<td>-ed ↔ Ø</td>
<td>Addition/ removal of simple past -ed</td>
</tr>
<tr>
<td>-ing ↔ Ø</td>
<td>Addition/ removal of gerund</td>
</tr>
<tr>
<td>for ↔ to</td>
<td>Preposition switching</td>
</tr>
<tr>
<td>for ↔ from</td>
<td>Preposition switching</td>
</tr>
<tr>
<td>from ↔ to</td>
<td>Preposition switching</td>
</tr>
<tr>
<td>at ↔ to</td>
<td>Preposition switching</td>
</tr>
<tr>
<td>through ↔ to</td>
<td>Preposition switching</td>
</tr>
</tbody>
</table>

Overall, errors in the 50 sentences from method 2 were principally one-to-one word errors and most often from the same word class. That is to say, the observed errors tended to consist of a word being mistaken for another word of the same word class.

The results were similar for the method 1 errors; certain single words were sometimes mistakenly transcribed as a sequence of multiple words or vice versa. It was also very common for one-to-one word errors of the same word class to be present. Given this, some assumptions will be made when we reach the error injection stage (see section 3.2.5).

**Grammatical errors:**

All of the notable grammatical errors have been generalised and are exemplified in Table 3.1. Many of the observed errors relate to verb tense; that is, there was a tendency for the ASR system to confound tenses and, for instance, add or remove morphological inflections in an incorrect manner. Furthermore, there was an observed trend for the system to switch articles, for example, from ‘the’ to ‘a’; since this usually does not change a sentence’s meaning in a significant way, such a mistake is considered to be minor. There was also a trend for plural marking to be muddled, so that, for instance, ‘piece’ would become ‘pieces’ or vice versa. Furthermore, there was a distinct tendency for prepositions to be mistaken for one another, as can be seen in Table 3.1.

**Phonetically-similar word errors:**

The most notable source of errors in the resulting transcripts of both method 1 and method 2 was the confounding of phonetically similar words. Words with the same number of syllables and similar phonetic structures were the most commonly affected. Other than some of the grammatical errors outlined in Table 3.1, there were no apparent patterns as to which words were confused. The likelihood of a word being misinterpreted depended heavily on level of noise in the recording, the sex of the speaker, and the context, among other factors.
3.2.5 Error creation and addition

The aim in injecting errors is to accurately and realistically reflect the type and frequency of errors that one may observe in a standard ASR system (such as Google Speech-to-Text\footnote{https://cloud.google.com/speech-to-text/}). In the ideal scenario, recordings of the Gigaword corpus articles would have been made and then fed into an ASR system for analysis, however, given the sheer size of the corpus and the timescale of this project, carrying out such tasks was unrealistic. Thus, the observations and analyses made in section 3.2.4 — along with some fair assumptions — will be made with regards to how to efficiently replicate realistic ASR errors.

Assumptions

Firstly, since the analysis in section 3.2.4 showed that the most common type of error was one-to-one word errors of the same word class, this is precisely the type of error that is used to imitate real errors. Secondly, given the phonetic nature of ASR, a system based on phonetic similarity is developed in order to replace words for phonetically similar (to varying degrees) words; this preserves the notion of an acoustic model within an ASR system. That is, acoustic models decode audio signals into phonetic representations and so by artificially altering these phonetic representations, we can imitate the errors that an ASR system may make. As is shown later in this section, many of the observed errors that are exemplified in Table 3.1 are also taken care of in this phonetic similarity system.

In order to take care of the prepositional errors, a separate system is utilised to make random prepositions switches (again, to varying degrees) for other prepositions to reflect common preposition mismatch pairs (as shown in Table 3.1). Five separate noisy versions of the Gigaword corpus are produced using these methods, ranging from mildly noisy (level 1) to extremely noisy (level 5) in addition to the baseline dataset with zero noise (level 0). Precise details on this noise-level system are available in Table 3.2.

Approach

In order to have a range of different levels of errors, we created five separate Gigaword corpus variations, each with a different level of noise, along with a baseline corpus (also known as ‘level 0’), consisting of the original Gigaword corpus. Level 1 has the fewest errors after level 0, and the number of errors progressively increases until reaching a maximum in level 5. This resulted in a total of 6 variations of the Gigaword corpus upon which training of text summarisation systems could take place.

Given the results of the noise analysis (see section 3.2.4), a system was devised to reproduce the observed errors across the entire Gigaword corpus in the levels of noise described above. This involved two separate stages: stage 1, introducing phonetic word errors, and stage 2, introducing prepositional word errors.

For stage 2, a rule-based approach was adopted. A percentage of the total articles in the training, validation and test sets would be targeted and, within each targeted article, a preposition would be replaced with another preposition. Which
preposition any given preposition could be replaced with depended on a set of
rules, which were themselves derived from the noise analysis in section 3.2.4. The
percentage of articles changed depended on the level of noise; this is illustrated in
Table 3.2.

For stage 1, the introduction of phonetic word errors, a more complex approach
was taken. The steps involved in this are detailed in the remainder of this section.
In order to inject phonetic errors into the Gigaword corpus, we switched words
within articles for phonetically-similar alternatives. For each variation of the
Gigaword corpus, a certain number of words were replaced with phonetically-
similar alternatives; the exact number of replaced words depended on the level of
noise, as is shown in Table 3.2. This process involved the creation of a pipeline
that would first create phonetically-based word vectors for a large sample of
the Gigaword corpus; this process is detailed in Figure 3.2. These vectors allow
for phonetically-similar words to be close in proximity within the vector space,
meaning that phonetically-similar alternatives to a queried word can be easily
located. This allows for words within the Gigaword corpus to be swapped for
realistic alternatives with both ease and accuracy, aptly reflecting the type of errors
one may find in ASR.

While this approach clearly takes care of the phonetically-similar word errors
detailed in section 3.2.4, it also deals with many of the grammar-based errors
noted from the noise analysis in the same section. For example, within the word
vectors, a queried verb will in most cases return different lemmas of the same root
verb as phonetically-similar alternatives; that is, querying ‘walk’ is likely to return
‘walked’ and ‘walking’ amongst its top similar results. Similarly, a queried singular
noun will most likely return its plural form and vice versa; that is, querying ‘boat’
will most likely return ‘boats’ amongst its top results. This means that, by using the
phonetically-similar word vectors, both grammatical and phonetic-based errors
are simulated, in an attempt to maximise the realism of the ASR errors.

<table>
<thead>
<tr>
<th>Level of noise</th>
<th>Phonetic errors / article</th>
<th>% prepositional errors / total prepositions in dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>70</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>90</td>
</tr>
</tbody>
</table>

Table 3.2: Distribution of errors in each dataset at each error level. Note that level 0 is the baseline, consisting of totally clean training data.
### Table 3.3: Word class dictionaries and the number of individual words they contained for the phonetically-similar word switching segment of this paper

<table>
<thead>
<tr>
<th>Word class</th>
<th>Number of words</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adjective</td>
<td>5080</td>
</tr>
<tr>
<td>Noun</td>
<td>18967</td>
</tr>
<tr>
<td>Verb</td>
<td>7749</td>
</tr>
</tbody>
</table>

In order to create phonetically-similar word vectors, firstly, we took a random sample of the Gigaword corpus articles (750,000 full articles). This random sample method was selected since it was deemed large enough to collect data on an extensive number and range of different words without expending too many computational resources.

Of this random sample of Gigaword articles, we generated part-of-speech (POS) tags for each word using SpaCy\(^8\) (Honnibal and Montani, 2017). We then stored each word and its corresponding POS tag in a dictionary containing only words of the same word class. Note that only dictionaries of nouns, verbs and adjectives were used; other word classes were considered to be too sparse to be of any true value in this exercise. Furthermore, we only use words with a frequency of 15 or more; this aided the process of filtering out uncommon words and words that

---

\(^8\)https://spacy.io/
had been misclassified by the POS tagger. Table 3.3 shows the three word class dictionaries along with their respective total numbers of unique words.

**Grapheme to Phoneme Conversion**

In the next step, phonetic representations were created for each word using a grapheme to phoneme converter\(^9\) which produces representations based on, and formatted in accordance to, the Carnegie Mellon University (CMU) Pronouncing Dictionary\(^10\) (Weide, 1998). The CMU style of phonetic representation stores, for example, the word ‘blue’ as ‘B L UW1’, and ‘America’ as ‘AH0 M EH1 R AH0 K AH0’, taking each individual grapheme and converting it to a phoneme. These phonetic representations were added to the dictionary entry for each word.

**Phonetically-similar representation of vectors**

The three dictionaries (one each for adjectives, nouns and verbs) were then fed to a phonetic similarity vector system\(^11\) (Parrish, 2017) in order to create word vectors based on each word’s phonetic representation. The system creates a vector space in which phonetically similar words have similar vectors. As the author writes, “[i]t must hold that for observations b1 and b2, the result of similarity (Prp(b1), Prp(b2)) increases according to the phonetic similarity of b1 and b2.” (Parrish, 2017). It functions by firstly converting each word to an Arpabet phonetic representation. Arpabet is a set of phonetic transcription codes developed by the Advanced Research Projects Agency (ARPA) (Rabiner and Juang, 1993). We then take phonetic bigrams of each word to calculate a comprehensive phonetic representation of the word. This is calculated by the function \( Prp \), where \( b \) is a word of length \( n \), \( p_i \) is a phoneme at position \( i \) and \( F(i) \) evaluates to a set of phonetic features that correlate to \( p_i \):

\[
Prp(b) = (F(p_1) \times F(p_2)) \cup \cdots \cup (F(p_{n-1}) \times F(p_n))
\]

After this has been created for each word in the vocabulary, a vector space is created based on similarity between word representations.

**Annoy**

In the next step, the information encoded in the previously-made vectors is used in conjunction with Annoy\(^12\) (Bernhardsson, 2018). Annoy — or Approximate Nearest Neighbor Oh Yeah — is a library that allows for extremely fast and efficient nearest neighbour searches, allowing users to determine the nearest neighbours to any given point in sublinear time. As input, Annoy takes a set of vectors. It then divides the vector space by randomly selecting two points and splitting them with a hyperplane that is equidistant to them, creating two distinct nodes. This splitting occurs recursively until there are a maximum of \( k \) points left.

---

\(^9\)https://github.com/Kyubyong/g2p
\(^10\)https://www.speech.cs.cmu.edu/cgi-bin/cmudict
\(^11\)https://github.com/aparrish/phonetic-similarity-vectors
\(^12\)https://github.com/spotify/annoy
in each node. The final result of an Annoy split vector space where $k=10$ can be seen in Figure 3.3.

![Figure 3.3: A complete Annoy representation of a vector space after recursive splitting where $k=10$. (Bernhardsson, 2015)](image)

The final result of this recursive splitting is a binary tree representation of the divided space. When a point is queried, the binary tree is traversed in order to locate its $k$ nearest neighbour.

In Annoy, “points that are close to each other in the space are more likely to be close to each other in the tree” (Bernhardsson, 2015). This is a result of the fact that close proximity between two points equates to a reduced probability that they will be split by a hyperplane; this is the foundation of Annoy. It allows for the phonetically-similar neighbours of a queried word to be determined quickly and efficiently, resulting in minimal computation time when replacing words within the Gigaword corpus for phonetically-similar alternatives.

**Injection of errors**

By using the vectors and Annoy, we were able to replace a number of words within a given sentence in the original Gigaword dataset with a phonetically-similar word of the same word class. Firstly, a list of 7 phonetically-similar words for any given word was generated by using Annoy, and then one of those words was randomly chosen as a replacement for the selected word. After some preliminary testing, we decided that, given the size of the input, 7 was the best size for similar words for allowing a variety of candidates while also preventing vastly dissimilar words.
Since the vectors for adjectives, nouns and verbs were already stored separately, a check was implemented such that words could only be replaced by a word of the same word class; this ensured grammatical integrity and imitated the workings of a strong language model in an ASR system. The number of switches made was dependent on the desired level of noise (see Table 3.2).

In a very small number of cases, carrying out the error injection stages outlined above did not work. This was mainly due to there being no words with phonetically-similar alternatives (as per stage 1) or no valid prepositions to replace (as per stage 2). While such failures were rare in comparison to the overall size of the Gigaword corpus, we decided that the integrity of the data must be contained. Thus, in cases where articles ‘failed’ in a dataset, the article (along with its corresponding summary) was removed completely from every dataset. This was done to ensure that all datasets across all levels were exactly the same length and had the same content in order to allow for comparability at the results stage. Thus, every noise level contained exactly the same sentences in exactly the same order, just with varying levels of noise. The final sizes of datasets are shown below:

- Training: 3,621,009
- Validation: 179,840
- Test: 189,493

We thus proceed to the summarisation stage with six different datasets, all with the above sizes and with varying noise levels.
4 Text summarisation system

4.1 Fairseq: a Seq2Seq approach to text summarisation

In this paper, we treat text summarisation as a neural machine translation (NMT) task. In this sense, we look at the task as translating from the source (the articles) to the target (the titles) and thus take a sequence-to-sequence (seq2seq) approach. Seq2seq learning, first introduced by Sutskever et al. (2014) for Google, converts one sequence (the source) into another (the target), taking into account both individual words and their context, and, traditionally, making use of RNNs. Since its conception, it has seen state-of-the-art results in NMT applications (Cho et al., 2014; Sutskever et al., 2014; Wu et al., 2016). Seq2seq systems consist of two main components: an encoder and a decoder: the first “encodes a sequence of symbols into a fixed length vector representation, and the other decodes the representation into another sequence of symbols” (Cho et al., 2014). In recent years, seq2seq modelling has had great success in the area of text summarisation (Nallapati et al., 2016; Rush et al., 2015; Shi et al., 2018), taking on many of the same principles as used in NMT. The only superficial difference is that instead of having source and target data in different languages as in NMT, in text summarisation the target data is simply a summary of the source data.

We make use of Fairseq\(^1\) (Gehring et al., 2016, 2017), a seq2seq modeling toolkit developed by Facebook AI Research with machine translation and text summarisation tasks in mind. It is built using PyTorch\(^2\) (Paszke et al., 2017). Fairseq is built upon a convolutional network encoder-decoder architecture with soft attention. One of the advantages of using convolutional neural networks for an application such as text summarisation is that they “do not depend on the computations of the previous time step and therefore allow parallelization over every element in a sequence” (Gehring et al., 2017), as opposed to RNNs which, due to their retention of past hidden states, intrinsically disallow parallel computation. Fairseq has previously best out several strong RNN-based baselines in the area of machine translation (Gehring et al., 2016, 2017). The convolutional structure of the system allows it to incrementally build context as well as a hierarchical understanding of the encoded sequence.

In this system, an encoder takes a sequence of \(m\) input words \(x=(x_1, ..., x_m)\) and processes it with a set of convolutions. With each convolution, more high-level features are learned and encoded. A gating system, composed of gated linear unit (GLU), is able to selectively filter information to feed up the hierarchy (Auli et al., 2018). As output, a sequence of states \(z=(z_1, ..., z_m)\) is given.

The translation component of the Fairseq calculates a distribution over \(V\) possible

\(^1\)https://github.com/pytorch/fairseq
\(^2\)https://github.com/pytorch/pytorch
target words

\[ p(y_i + 1|y_1, ..., y_i, x) = \text{softmax}(W_0h_i + 1 + b_o) \]

where \( y_i + 1 \) is the next possible target word, \( h_i \) is the LSTM encoder representation output, \( W_0 \) is the weights and \( b_o \) is the bias.

A conditional input, \( c_i \) is then computed by taking a weighted sum of attention scores \( a_i \in \mathbb{R}^m \) from each time step. Each attention score \( a_i \) is calculated as the dot product between \( h_i \) (decoder context representation) and \( z_j \) (encoder representation). These steps culminate in the prediction and production of a final output in the target language.

The architecture of this entire process is illustrated in Figure 4.1.

---

4.2 Hyperparameters

Once all datasets had been binarised, the training stage could commence. This began with the tuning of two hyperparameters: the optimiser and the learning
rate.
In order to tune the hyperparameters, a semi-greedy search was implemented. All tuning was carried out on the level 0 (clean) version of the Gigaword corpus. While this version had zero noise, we decided that it would be fairest to tune hyperparameters on since this data serves as the baseline for all of the experiments in this paper.
All efforts were taken to replicate the training environment used in the main experiments as closely as possible. However, only level 0 validation data was used to evaluate the models. This was done in order to conserve both time and resources. We use ROUGE metrics to evaluate the models; more information on these metrics can be found in section 5.1. Other than this, every training aspect remained the same. Full specifications of the training environment may be found in section 4.2.4.

4.2.1 Optimiser
Firstly, the optimiser was tuned. Optimisers are a key component of machine learning and work to update the model in relation to the loss function with the overall aim of making the model as accurate as possible. As Bahar et al. (2017) notes, “[f]ast convergence and robustness against stochasticity are important aspects desired in an optimizer so that it finds the global optimum.” We trained five Fairseq models, each with a different optimiser. The optimisers were as follows:

- Adam
- Adadelta
- Adagrad
- Nesterov accelerated gradient (NAG)
- Stochastic Gradient Descent (SGD)

At this stage, the default Fairseq learning rate setting (0.25) as well as all of Fairseq’s other default settings were employed. The results can be seen in Table ?? and Figure ??.
As may be seen from Figure 4.2, it was discovered that the best optimiser for this system was the NAG optimiser (Nesterov, 1983). It must be noted, however, that NAG was only marginally better than SGD. NAG is used as the default optimiser in Fairseq training as it has been found that it is well suited to applications of machine translation and text summarisation (Gehring et al., 2017). It is defined by Sutskever (2013) as “a first-order optimization method which is proven to have a better convergence rate guarantee than gradient descent for general convex functions with Lipschitz-continuous derivatives ($O(1/T^2)$ versus $O(1/T)$)” (as seen in classic gradient descent). NAG is considered to be closely related to momentum methods (see Sutskever et al. (2013) and Qian (1999)),

24
Table 4.2: Results of optimiser tuning. In the table (left), ROUGE scores are given as the mean of the F-1 ROUGE-1 score and the F-1 ROUGE-2 score. Best scores are shown in bold. In the figure (right), results are given as the F-1 ROUGE-1, F-1 ROUGE-2 and F-1 ROUGE-L scores.

<table>
<thead>
<tr>
<th>Optimiser</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adam</td>
<td>0.43</td>
<td>0.219</td>
<td>0.387</td>
</tr>
<tr>
<td>Adadelta</td>
<td>0.421</td>
<td>0.201</td>
<td>0.378</td>
</tr>
<tr>
<td>Adagrad</td>
<td>0.375</td>
<td>0.167</td>
<td>0.344</td>
</tr>
<tr>
<td>NAG</td>
<td><strong>0.441</strong></td>
<td><strong>0.264</strong></td>
<td><strong>0.404</strong></td>
</tr>
<tr>
<td>SGD</td>
<td>0.435</td>
<td>0.229</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Figure 4.2: Results of optimiser tuning. In the table (left), ROUGE scores are given as the mean of the F-1 ROUGE-1 score and the F-1 ROUGE-2 score. Best scores are shown in bold. In the figure (right), results are given as the F-1 ROUGE-1, F-1 ROUGE-2 and F-1 ROUGE-L scores.

with the only difference being the exact update of the velocity vector, $v_t$. The precise update algorithm used in NAG may be written as (Ruder, 2016):

$$
v_t = \gamma v_{t-1} + \eta \nabla\theta J(\theta - \gamma v_{t-1})
$$

$$
\theta = \theta - v_t
$$

In the above equation, we take $\gamma v_{t-1}$ as our momentum term to adjust our parameters ($\theta$). We can then calculate the approximate our parameters’ next position by evaluating $\theta - \gamma v_{t-1}$.

In simpler terms, the notion of this type of optimisation is aptly described by Ruder (2016): “a ball that rolls down a hill, blindly following the slope, is highly unsatisfactory. We would like to have a smarter ball, a ball that has a notion of where it is going so that it knows to slow down before the hill slopes up again” — this is NAG optimisation.

4.2.2 Learning Rate

Taking NAG as the optimiser, as seen in 4.2.1, we proceed to the tuning of the learning rate. The learning rate controls the step with which weights are adjusted in the model. Having an optimal learning rate for a particular model is crucial as having too small of a learning rate may result in never reaching the ideal one, while having too large of a learning rate risks overstepping and completely missing the best one. Note that by default, however, Fairseq reduces the learning rate as it begins to plateau, thus reducing learning steps as it reaches convergence. This approach is useful in minimising the risk of overstepping.

We tested a range of learning rates from 0.000001 to 1 (inclusive), using logarithmic steps as intervals. After analysing the initial results and discovering that a learning rate of 0.25 gave the best results, two additional experiments were run with learning rates of 0.2 and 0.3, respectively, in order to fine tune the learning rate further. All results are demonstrated in Figure 4.3.

As Figure 4.3 shows, the optimal learning rate was 0.3. This is close to the Fairseq default learning rate of 0.25.
4.2.3 Final hyperparameters

Given the results of the hyperparameter tuning in section 4.2, the all six noise versions of the Gigaword corpus data were trained with:

- Optimiser: NAG
- Learning rate: 0.3

For the remaining hyperparameters that the Fairseq system requires, Fairseq’s default settings were used. These were as follows:

- Dropout: 0.2
- Maximum number of tokens per batch: 4000
- Model architecture: fconv\(^3\)
- Clip threshold of gradients: 25
- Momentum factor: 0.99
- Weight decay: 0
- Learning rate shrink factor: 0.1
- Minimum learning rate: 1e-05
- Training criterion: Cross entropy
- Required batch size multiple: 8

\(^3\)Fairseq’s architectures determine the network configuration, for instance such aspects as embedding dimension. \texttt{fconv} is the default setting and signifies a fully convolutional encoder and decoder. It uses an embedding size of 512.
Given that the tuning of both optimiser and learning rate ended up yielding optimal settings that were either close or identical to Fairseq’s default settings, there was reasonable evidence to suggest that this may also be the case for dropout. Thus, we used Fairseq’s default setting for dropout (0.2).

The models were not trained for a set number of epochs; instead, we used Fairseq’s learning rate shrink factor to allow the models to naturally converge. All six final models trained for between 30 and 33 epochs, inclusive. This lack of variation stems from the fact that all six models were trained on datasets of equal size, the only difference being the noise level.

Full details of Fairseq’s default settings may be found here⁴.

4.2.4 Training environment

All models were trained and tested on an IBM Power System AC922 server with four NVIDIA Tesla V100 Tensor Core GPUs. The server consists of two POWER9 CPUs and has a CentOS7 Linux distribution.

⁴https://fairseq.readthedocs.io/en/latest/command_line_tools.html#fairseq-train
5 Environment

5.1 Evaluation method

The evaluation of text summarisation models has posed problems for many researchers. The subjectivity of the outputs as well as the infinite range of outcomes makes for difficulties when designing evaluation metrics. The most popular evaluation metrics at the time of writing (Lin, 2004) rely on a comparison between a system summary and a reference summary, however, as Steinberger and Jezek (2009) notes “[t]he problem of matching the system summary against an ‘ideal summary’ is that the ideal summary is hard to establish”.

A decent evaluation metric for text summarisation must take many factors into account, including relevance, coherence and conciseness. Despite the intrinsic difficulties in evaluating text summarisation models, we must select a metric. We opt for ROUGE\(^1\) (Lin, 2004). ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation and is a set of metrics designed specifically with automatic text summarisation in mind. Over the last decade, ROUGE has become the industry standard and the most widely-used evaluation metric for text summarisation tasks.

We also explore the idea of using ROUGE 2.0\(^2\) (Ganesan, 2018) which addresses issues present in the first version of ROUGE. One of the most pertinent to the use in this paper is that the original implementation of ROUGE does not take into account synonymous concepts. Even between human-produced summaries, it is perfectly normal and accepted for synonyms to be used to describe identical ideas. However, ROUGE fails to recognise this and considers any given synonym to a word in the reference summary to be erroneous. This issue is illustrated in the example below:

- **REFERENCE**: The boy sat on the sofa.
- **SYSTEM**: The boy sat on the couch.

In the example above, the system summary would be penalised as per the original ROUGE metrics since it uses the word ‘couch’ instead of ‘sofa’. ROUGE 2.0 resolves this by using a synonym look-up dictionary to search for accepted synonyms, thus giving equal treatment to reference words as well as their synonyms. This means that in the example above, the system summary would receive a perfect score since ‘couch’ is considered to be synonymous with ‘sofa’. This ultimately leads to a more reliable and realistic evaluation of summary similarity, as shown by Ganesan (2018).

Both ROUGE and ROUGE 2.0 include the same basic metrics, but vary slightly in

\(^1\)https://github.com/pltrdy/rouge
\(^2\)https://github.com/kavgan/
approach. The main differences are the addition of a synonym lookup in ROUGE 2.0, as explained above, as well as the removal of stopwords. The aim of the two systems is essentially the same, however, to compare a system (or 'machine output') summary with a reference summary (or set of reference summaries), usually created by humans. The range of metrics includes ROUGE-N, which allows for a measure of N-gram overlap between the summaries, ROUGE-L, which measures the longest matching sequence between summaries, and ROUGE-S, which measures skip-gram concurrence. To further exemplify this, let us take an example of how a unigram (ROUGE-1) measure would function. Imagine that the following is a human-produced reference summary:

The child played in the garden.

We may then imagine that a machine could produce a summary such as:

The happy child was playing in the garden.

A recall score may then be calculated from this as the number of overlapping words between the two summaries over the total number of words in the system summary. In this case:

\[
\frac{\text{number of overlapping words}}{\text{total number of words in reference summary}} = \frac{5}{6} = 0.83
\]

While this metric gives some notion as to the quality of the summary, it misses out a crucial fact: it does not check the length of the summary. Since the goal of text summarisation is to produce a concise version of a given text, it seems natural that there should be at least some control to ensure that the summary is not overly lengthy or does not contain too many irrelevant words. Therefore, we also take a precision measure. This evaluates the number of overlapping words over the number of words in the system summary. Taking the same example, this gives:

\[
\frac{\text{number of overlapping words}}{\text{total number of words in system summary}} = \frac{5}{8} = 0.63
\]

Since both precision and recall are important in gauging the quality of a summary, one may take the F-measure when analysing summaries in order to take both factors into account. While these examples illustrate how ROUGE's most basic metric, ROUGE-1, functions, it must be said that some of the other ROUGE metrics are more intuitive when it comes to text summarisation. For instance, looking at bigram overlap, which is done in ROUGE-2, takes into account word order — which naturally tends to relate to a coherent summary — rather than the bag-of-words approach seen in ROUGE-1. Since it is both problematic and unreliable to gather conclusive results from one single metric, we took three metrics instead, following Rush et al. (2015). The metrics are as follows:

**ROUGE-1**: As outlined above, ROUGE-1 tries to match unigrams from the reference summary to the system summary and awards points accordingly. While this metric gives no pointers as to the syntax or
overall intelligibility of a given system summary, it is important to
gauge general similarity to a reference summary.

**ROUGE-2:** ROUGE-2 functions in a similar way to ROUGE-1 except
that, as the name suggests, it instead matches bigrams. This aids in
retaining some sense of syntax and coherence.

**ROUGE-L:** ROUGE-L calculates the longest comment subsequence
between the reference and system summary. “Given two sequences
X and Y, the longest common subsequence (LCS) of X and Y is a
common subsequence with maximum length” (Lin, 2004).

Our evaluation system (ROUGE) uses all of these metrics and gives their individual
F-1 scores. Note that while we originally evaluated all models using ROUGE 2.0,
we realised post-experiments that there was an unclear issue with this system
that was causing both computational problems and peculiar results patterns. We
therefore decided to switch evaluation systems and to retest everything with the
original ROUGE metrics. Although this is an unconventional choice, we feel that it
was justified as it threatened the integrity of our results. Furthermore, our use of
ROUGE allows us to more readily compare our results to those of state-of-the-art
papers since they all use this system.

### 5.2 Goals and hypotheses

In the remainder of this paper, we take *noiseless model/data* to refer to the level 0
model/data, *low-noise models/data* to refer to the level 1 and 2 models/data, and
*high-noise models/data* to mean the level 3, 4 and 5 models/data.

First of all, the overall aim of these experiments is to create a noise-robust
text summarisation system. This aim is encouraged by training neural networks on
text-summary pairs in which the text has a varying level of noise (outlined in sec-
tion 3.2.5) but in which the summary is always completely clean (i.e. noise-free).
While the general goal of creating noise-robust text summarisation models is
within reach, we do not expect exceptional results due to the small scale and
scope of this project, especially given the notorious difficulty surrounding text
summarisation. However, we hope that there will be meaningful contributions
and steps in the right direction towards creating such systems.

For the final experiments, we have the following hypotheses:

**Hypothesis #1**

We hypothesise that noisy models will be better adapted to summarising noisy
data. We suspect that, since the noiseless and low-noise models have been trained
on no or little noisy data, they will perform poorly when summarising high-noise
data.
Hypothesis #2

We hypothesise that the baseline model (noiseless model) tested on noiseless data will score best.

Hypothesis #3

Finally, we hypothesise that the baseline model (noiseless model) tested on level 5 data will perform the most poorly.
6 Results

All models were tested on each of the test datasets of the Gigaword corpus. In total there were 6 different test datasets, all consisting of the same 189,493 articles-summary pairs, but varying from noise levels 0-5, inclusive. As outlined in section 5.1, ROUGE-1, ROUGE-2 and ROUGE-L metrics were used to evaluate all models. The results of testing these are shown in Table 6.1 as individual results for the F-score of each. The scores in italics represent those of models tested on the same noise level of data as they were trained on. Underlined results signify the overall best score for each ROUGE metric.

Within the results, there is a tendency for all three ROUGE metrics at each test instance to fluctuate proportionally. Therefore, for reasons of clarity, we take the mean of all three ROUGE metrics for each instance to represent the final results in Table 6.1. Similarly, we show the final results of models tested on homogeneous data by giving their mean score across all three ROUGE metrics in Figure 6.2.

<table>
<thead>
<tr>
<th>MODEL</th>
<th>Test Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Level 0</td>
</tr>
<tr>
<td>Level 0</td>
<td>R1 0.444 R2 0.264 RL 0.409</td>
</tr>
<tr>
<td>Level 1</td>
<td>R1 0.103 R2 0.052 RL 0.098</td>
</tr>
<tr>
<td>Level 2</td>
<td>R1 0.079 R2 0.05 RL 0.07</td>
</tr>
<tr>
<td>Level 3</td>
<td>R1 0.059 R2 0.021 RL 0.049</td>
</tr>
<tr>
<td>Level 4</td>
<td>R1 0.052 R2 0.021 RL 0.059</td>
</tr>
<tr>
<td>Level 5</td>
<td>R1 0.052 R2 0.019 RL 0.042</td>
</tr>
</tbody>
</table>

Table 6.1: Final results of testing each of the 6 models (trained on varying noise levels) on each of the noise level’s test data. Individual results for the F-1 ROUGE-1 score (R1), F-1 ROUGE-2 score (R2) and F-1 ROUGE-L score (RL) are given. Results in italics represent those of models tested on homogeneous test data. Underlined results signify the overall best score for each ROUGE metric.
Figure 6.1: Final results of testing each of the 6 models (trained on varying noise levels) on each of the noise level's test data. Scores are given as the mean of the F-1 ROUGE-1(R1), F-1 ROUGE-2(R2) and F-1 ROUGE-L(RL).

Figure 6.2: Final results of testing each of the 6 models (trained on varying noise levels) on homogeneous data. Scores are given as the mean of the F-1 ROUGE-1 score and the F-1 ROUGE-2 score.
7 Discussion

7.1 Analysis of results

In this section, we refer to homogeneous datasets as those in which the noise level is the same as the data used to train the specified model, and non-homogeneous datasets as those in which the noise level differs from the data used to train the mentioned model.

From Table 6.1, we can see that the models tested on homogeneous test data (shown in italics) scored consistently higher than when tested on non-homogeneous test data. This is unsurprising since these models were trained on datasets with the same level of noise as they were tested on. What is interesting is the extreme drop in results when testing on non-homogeneous data.

We also note the fact that we had originally evaluated these models with the ROUGE 2.0 system as opposed to the ROUGE system used now (see section 5.1 for further details). We must remark that the only aspect that the ROUGE 2.0 results pattern had in common with the original ROUGE metric was the similarity between the results of models tested on homogeneous data.

7.1.1 Outcome of hypotheses

Hypothesis #1 stated that noisy models would be better adapted to summarising noisy data. This was the case to a certain extent, however, what we found was that, in reality, models tended to only perform above average when tested on homogeneous data — this was indicative of overfitting (see section 7.1.4 for more).

This meant that the level 5 model only summarised level 5 data well, and not, for example, level 4 data. Furthermore, it was also true that low-noise and noiseless models did perform poorly in summarising high-noise data, however, it cannot be said that they did a worse job than in summarising any other non-homogeneous data. It was true for all models that results from all non-homogeneous data had little variation.

Hypothesis #2 was partially confirmed: while the best mean score did come from model 0, it was tied with model 1; when tested on homogeneous data, they both achieved a score of 0.672. From level 1 to level 5, mean scores on homogeneous data decreased very marginally, showing that the models did decrease in quality ever so slightly as the noise level increased.

In fitting with hypothesis #3, the poorest mean score comes from model 0 applied to level 5 data. The model, which was trained on clean data, clearly struggled to decipher and summarise the noisy data.

The level 5 model proved itself to be apt at summarising homogeneous data, however, like all of the other models, it struggled with non-homogeneous data. The cause for this is unclear: all of the test data across all levels were derived from
the same source — they had simply had a different level of noise applied to them, as outlined in section 3.2.

7.1.2 Individual models

While we will observe all three of ROUGE-1, ROUGE-2 and ROUGE-L, to ease comparison and analysis we will focus primarily on ROUGE-L scores in this section.

As expected, the level 0 model performs best when tested on homogeneous data. When tested on non-homogeneous data, the scores decrease significantly, going from a ROUGE-L score of 0.409 on homogeneous data to a range of 0.018 to 0.149 on non-homogeneous data. We find that as the level of noise increases, model 0’s performance decreases. Overall, this model performed worst when applied to the noisiest dataset (level 5) compared to the other models.

For the level 1 model, when tested on homogeneous data, a ROUGE-L score of 0.408 is achieved. However, for non-homogeneous datasets, a score of between 0.028 and 0.098 was obtained. Of all models, excluding the level 0 one itself, level 1 does the best job of summarising level 0 data, garnering a ROUGE-L score of 0.098. The level 1 model’s performance decreases as the noise level of the test dataset diverges from its own.

The level 2 model yielded similar results to those of level 1. On non-homogeneous datasets, the model achieved ROUGE-L scores ranging from 0.027 to 0.076. On homogeneous data, however, a ROUGE-L score of 0.405 was obtained.

The level 3 model also produced comparable results to those of levels 1 and 2. Its ROUGE-L score on homogeneous data was 0.403. The ROUGE-L scores on non-homogeneous data ranged from 0.045 to 0.049. Here, we see the score on level 5 data increasing slightly as the score on level 0 data decreases, showing that resistance to noise is being built — at least to some extent — as performance on clean data weakens. This is the first model in which we see scores on level 5 data that exceeds those on level 0 data.

The level 4 model, as with the other models, when tested on homogeneous data produced its highest score: 0.4 (ROUGE-L). The ROUGE-L scores on non-homogeneous data had a range of 0.039 to 0.069. Once again, resistance to noisy data seems to be increasing slightly.

Finally, the level 5 model also performed highest when tested on homogeneous data, as expected. Here, it garnered a ROUGE-L score of 0.399. For non-homogeneous datasets, ROUGE-L scores ranged from 0.042 to 0.079. We see a trend that as noise increases, so too does the model’s performance. This shows both resistance to some noise and a growing weakness to clean data.

Finally, Figure 6.2 shows that the models’ aptitude to summarise homogeneous data did decrease slightly as the level of noise increased. This is likely due to the obvious fact that the data was arbitrarily noisy, leading to a larger margin of error when summarising unseen data. However, overall, there was only a relatively small decrease in performance — 0.004 from level 1 to level 5 — showing that even the noisiest model was able to maintain many of the abilities of the baseline model.
7.1.3 Overall

Non-homogeneous scores

We first discuss the inconsistencies and peculiarities with our results. One of the most striking aspects of the results is the sheer difference in homogeneous and non-homogeneous results, made more apparent by Table 6.1. Although there is a slight variation between models in terms of the exact scores, the trend that we observe remains more or less the same: a relatively high score on models tested on homogeneous data and then a sheer drop when applied to non-homogeneous data. When considering the mean of ROUGE-1, ROUGE-2 and ROUGE-L scores, we witness an average drop of 86% from results on homogeneous data to those on non-homogeneous data. Such a difference is quite unexpected and points towards problems somewhere along the pipeline. We carried out some sanity checks in order to determine the exact cause of the issues. To do so, we divided the level 0 and level 5 test sets into 5 equal parts and tested the level 0 and level 5 models on each of them and then averaged out the results. What we achieved were exactly the same results as when the models were tested on the equivalent whole datasets, within a margin of 0.5% accuracy. This demonstrates that the source of error was not to be found within the evaluation stage of the pipeline. The results are indicative of overfitting, however: we draw this conclusion since the models are particularly good at summarising homogeneous data but the score drops down so dramatically when the level of noise in the test set is just one level different from that of the training set.

However, while we do see a large drop from homogeneous to non-homogeneous results, we also see the trend for scores to decrease as the level of noise in the test data verges away from that on which the model was trained. This was to be expected: the more the test data resembles the training data, the better we predict the model to perform, and vice versa. This trend is illustrated in Figure 6.1.

In order to further test this observation, we created an additional three noisy datasets, this time with noise levels of 8, 11 and 14. We tested all six of our existing models on these datasets. The results of this are shown in Figure 7.1. In this graph, scores of zero are represented as black dashes for reasons of clarity.

We see that the level 5 model is the only one that is able to obtain a non-zero result on all three of the additional noisy test sets. Furthermore, as expected, a downward trend is maintained as the noise level increases. Level 4 achieves non-zero scores on the level 8 and level 11 test sets, but no the level 14 one. Once again, a downward trend is observed as the noise level increases. The remaining four models achieve scores of zero on all of the additional three test sets. This signifies that these models have failed and that the level of noise has overpowered their ability to summarise. Similarly, we see model 4 fail from the level 11 to the level 14 dataset. Whilst model 5 proves itself to be the most robust to noise with at least some ability to summarise the level 15 test data, the clear downward trend implies that it too will fail at some point with the addition of even more noise.

Homogeneous scores

Another observation that we make in the results is that all of the homogeneous results that we report are slightly higher than the state-of-the-art models; for
Figure 7.1: Final results of testing each of the 6 models (trained on varying noise levels) on non-homogeneous test data. This includes the addition of three new datasets with noise levels of 8, 11 and 14. Black dashes represent scores of zero. Scores are calculated as the mean of the F-1 ROUGE-1(R1), F-1 ROUGE-2 (R2) and F-1 ROUGE-L (RL) scores.

instance, we obtain a ROUGE-L homogeneous result on our level 0 model of 0.409 and the equivalent on our noisiest model (level 5) of 0.399. The top three ROUGE-L scores obtained so far for the task of text summarisation are 0.3775, 0.377 and 376 (Al-Sabahi et al., 2018; Jadhav and Rajan, 2018; Wu and Hu, 2018). One must also bear in mind that these state-of-the-art models have been trained and tested on clean — not noisy — data. The Fairseq toolkit has performed very well in sequence-to-sequence tasks (Fan et al., 2017; Ott et al., 2019) and while the results we report are not exceptionally higher than the state-of-the-art scores, we suspect that our models are, in fact, overfitting. We explore this by cross-referencing our training and test data for some area of overlap. What we discover is that within the original Gigaword corpus — prior to splitting into training, validation and test sets and the addition of noise — there were ∼300,000 instances of repeated article-summary pairs out of a total of 4 million. This is a compelling reason to expect a degree of overfitting in our models’ performance, especially if identical sentences appear in both the training and test sets. However, while this likely leads to some overfitting, we must bear in mind that for those models with added noise, the noise is added in an arbitrary way and so, even for cases of duplicate articles, after the addition of noise they will likely vary to some degree.

Rouge metrics on noisy data

We must also attribute a portion of the blame to the performance of our evaluation system, ROUGE, in combination with the addition of noise on the datasets. Since we know that the level of noise is based on word-level noise, it is plausible to
imagine that such noise could quite easily affect our metrics’ outcome. To illustrate this, let us take the following simplified example:

**Reference summary:** I just wanted to *be* happy in my life.
**System summary:** I just wanted to *see* happy in my life.

In the above example, we see the longest common substring (LCS) of the system summary in relation to the reference summary would be four words — either ‘I just wanted to’ or ‘happy in my life’. Here, the switching of just one word, *be* to *see* — that is, the addition of one noise level — leads to a significant decrease in ROUGE-L score due to the fact that the LCS is just four out a possible maximum of nine. This demonstrates how even a single error can cause considerable decreases in ROUGE evaluations.

A similar, but less extreme, conclusion can be drawn for ROUGE-2 results. A model tested on just one or two noise levels higher than the data that it was trained upon will naturally be more prone to potentially significant decreases in bigram matches. In the above example, there is a reduction in bigram matches of 25%, just from one word having been changed. This therefore leads to a lowered ROUGE-2 score.

While ROUGE metrics are the most widely-used text summarisation evaluation system, the evidence outlined here highlights reasons for which one must be wary when applying it to noisy text summarisers.

### 7.1.4 Result examples

In this section, we take a closer look at some of the output from the text summarisation models. We look at one example on noiseless data (level 0) and two on very noisy data (level 5), and each time looking specifically at the output generated by all six models. The articles shown in the tables below represent what was presented to each of the six models for summarisation.

In order to keep this evaluation fair, all articles were selected prior to the data being generated.

<table>
<thead>
<tr>
<th>ARTICLE: Mike Piazza’s left elbow was still ‘a little stiff’</th>
<th>REFERENCE SUMMARY: Piazza overcomes elbow stiffness</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Model</strong></td>
<td><strong>System summary</strong></td>
</tr>
<tr>
<td>Level 0</td>
<td>Piazza’s elbow still a problem</td>
</tr>
<tr>
<td>Level 1</td>
<td>Piazza overcomes problems</td>
</tr>
<tr>
<td>Level 2</td>
<td>Piazza day-to-day with injury</td>
</tr>
<tr>
<td>Level 3</td>
<td>A look at what’s doing in Piazza</td>
</tr>
<tr>
<td>Level 4</td>
<td>Don’t feel the pain of the Piazza</td>
</tr>
<tr>
<td>Level 5</td>
<td>Not known is cause of stiffness</td>
</tr>
</tbody>
</table>

**Table 7.1:** Summaries generated on the level 0 test dataset.

In Table 7.1, the outputs of models 0, 1 and 2 provide apt summaries of the given article. They all include the subject (Piazza) as well as details of or references to his injury. The level 3 and level 4 outputs both also mention Piazza, however...
the level 3 output is grammatically and syntactically odd and the level 4 output conveys the wrong message. Finally, the level 5 model, which is trained only on noisy data, struggles to encode most of the information in the article. It does, however, manage to include a mention of the ‘stiffness’. This shows that, in this case, the level 5 model performs poorly in the summarisation of clean articles.

<table>
<thead>
<tr>
<th>Model</th>
<th>System summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 0</td>
<td>Eriksson and seaman don’t know what to do</td>
</tr>
<tr>
<td>Level 1</td>
<td>Eriksson says he’s satisfied with North Korean success</td>
</tr>
<tr>
<td>Level 2</td>
<td>Brazil’s &lt;unk&gt; meet with Brazil’s &lt;unk&gt;</td>
</tr>
<tr>
<td>Level 3</td>
<td>Burst of &lt;unk&gt; &lt;unk&gt; in Brazil</td>
</tr>
<tr>
<td>Level 4</td>
<td>Brazilian Olympians in Canada go on trial</td>
</tr>
<tr>
<td>Level 5</td>
<td>Eriksson says England on a par with Brazil in World Cup</td>
</tr>
</tbody>
</table>

Table 7.2: Summaries generated on the level 5 test dataset.

Firstly, it is important to realise that the original article in Table 7.2 was actually:

Sven-Goran Eriksson believes England are on a par with five-time champions Brazil and can win the World Cup as long as they stay clear of injuries, penalty shoot-outs, and get a little luck.

The article shown in Table 7.2 is the result of the above article having had level 5 noise applied to it. If we examine the level 5 article, we can see that there are grammatical errors — for instance, the use of ‘can will’ — and lexical issues — for example, ‘stating clear’ and ‘world puk’. The level 0 and level 1 summaries both manage to include ‘Eriksson’, however, the rest of their summaries have no connection to the original article. Similarly, the level 2 and 3 models perform poorly: they both overfit to the ‘Brazil’ aspect of the article and lose all of the other key information, resulting in inadequate summaries. The level 4 output falls prey to the ‘Olympians’ token, which was actually supposed to be ‘champions’ and includes this in its summary. The level 5 summary, however, does a good job of including many — if not all — of the key information, from ‘Eriksson’ to ‘England’ and ‘Brazil’ to ‘World Cup’. Not only this, but the summary also encodes the fact that England are ‘on par’ with Brazil and the fact that this was all said by Eriksson. Another key point is that the level 5 output is able to filter out the noise of the article; the summary is completely grammatical and coherent and does not fall into the trap of encoding the noise of the article. The level 5 output can thus be considered to be a well-formed and informative summary of the article.
US toymaker were dealt another bit of bad news Sunday as the prestigious North Mclaren car of the year award was presented for the wounded Civic and its jinping truck of the year honors waned to the Honda Ridgeline.

REFERENCE SUMMARY: Honda wins North American car truck of the year

<table>
<thead>
<tr>
<th>Model</th>
<th>System summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 0</td>
<td>&lt;unk&gt; world awards first &lt;unk&gt; egg</td>
</tr>
<tr>
<td>Level 1</td>
<td>Honda rolls out new roll roll roll</td>
</tr>
<tr>
<td>Level 2</td>
<td>Honda may have accidentally accidentally &lt;unk&gt;</td>
</tr>
<tr>
<td>Level 3</td>
<td>Milestone milestone for Honda’s &lt;unk&gt;</td>
</tr>
<tr>
<td>Level 4</td>
<td>Two Honda agenda Batman</td>
</tr>
<tr>
<td>Level 5</td>
<td>US automakers dealt bad news in Honda award</td>
</tr>
</tbody>
</table>

Table 7.3: Summaries generated on the level 5 test dataset.

In Table 7.3, the original article was:

US automakers were dealt another bit of bad news Sunday as the prestigious North American Car of the Year award was presented to the Honda Civic and its sibling Truck of the Year honors went to the Honda Ridgeline.

However, since the article is level 5, several artificial errors have been injected. From Table 7.3, we can see that the level 5 model has rather successfully captured the notion of the article, including both the US automakers (as opposed to ‘toymakers’, as described in the noisy article), Honda and the link between them. Furthermore, the level 5 model succeeds in filtering out the misinformation presented in the noisy article, for example, such noise as ‘Mclaren’ and ‘wounded’. The output from levels 1-4 all include mentions of ‘Honda’. The overall summaries, however, do not aptly summarise the given article, from mentions of Batman to repetitions of the word ‘roll’. The only aspect that the level 0 output and the article have in common is the word ‘award’. Other than that, the summariser has included the word ‘egg’ that has nothing to do with the article. In this case, then, the level 5 summariser aptly provides a noise-free summary of the given article. What is interesting to note, however, is the fact that the level 5 output achieved ROUGE-1, ROUGE-2 and ROUGE-L scores of 0.118, 0 and 0.116, respectively — this is considered low. This reason for this is that, according to the ROUGE metrics used, there is little overlap between the two summaries. Thus, although a human judge would likely deem the level 5 output to be sufficient or even excellent, it receives a poor score based on the ROUGE metrics. However, because of the metrics, the summary is in fact penalised. This issue is discussed further in section 7.2.

Robustness against noise

The results from the level 5 model (Table 6.1) show that we have created a text summarisation system that is robust to noise. The model was trained on article-summary pairs where the articles were highly noisy and the summary was clean. What the results in Table 6.1 and Figure 6.1 demonstrate is that model 5
has an aptitude for deciphering and summarising noisy data. It is clear from the table that the low noise models struggle to decipher the noise, given their poor summaries.

**Overfitting**

All of the models show a tendency to overfit to their own noise level. We have discussed this issue throughout the section. This is evident from the fact that all models perform significantly worse on non-homogeneous data, even when the data comes from a neighbouring noise level. This means that, even though the level 5 data performs particularly well on homogeneous data, and thus fulfils the main objective of this paper to create a noise-robust text summarisation system, the model still overfits too much and is too specific to its training data.

In this paper, the main goal of creating a noise-robust text summarisation system is to be able to summarise all ASR output; not just noisy output. Thus, the summariser must be both robust to noise and flexible. Therefore, there is a need to normalise the model to prevent it from overfitting to such an extent. This could potentially be done by training it on both (i) more data and (ii) data with a range of noise levels as opposed to only noisy data.

### 7.2 The problem with text summarisation evaluation

As previously discussed in 5.1, text summarisation is an intrinsically problematic process to evaluate. The issue lies in attempting to place measures on natural language solutions, especially when there are a possibly infinite number of different but correct summaries for any given article. ROUGE is presently the industry standard when it comes to text summarisation evaluation, however, even within this, we find that there are several issues. This section will outline a selection of these problems, illustrated by output from the models trained in section 6. Note that in the examples `<unk>` is indicative of an unknown token and replaces a digit.

**Correct, but different: False negatives**

This category of ROUGE issues is one of the most problematic, with much work going into mitigating its effects (Nenkova and McKeown, 2012). It is the case in which a summary is correct, but has a different structure and/or vocabulary compared to the reference summary. An example of this is given in Table 7.3. We also give another example below in which the system summary could well be considered to be a fully correct summary for the given article, however, other than the words ‘Irish’ and ‘famine’, there is not a lot of resemblance between the reference and system summary. Furthermore, the two crossover words are not adjacent, therefore meaning that the system summary will also lose out when it comes to the bigram metric. All in all the system summary will be scored rather low. We must also consider that it is highly probable that there are more extreme examples of false negatives; that is, examples in which all words between reference and system summaries are different, but in which the system summary is just as plausible as the reference summary. In these cases, a score of 0 will be
given to the system summary and it will be considered to be a complete failure.

**Article:** In May ####, at a ceremony in Cork, Ireland, commemorating those who suffered and died in the Great Irish Famine, which began in, prime minister Tony Blair of Britain said: “that million people should have died in what was then part of the richest and most powerful nation in the world is something that still causes pain as we reflect on it today.”

**Reference summary:** Sites remember the Irish potato famine.

**System summary:** A year after the Great Irish Famine.

The effect of false negatives in text summarisation evaluation can be potentially minimised by including several reference summaries as opposed to a single one (as is done in this project).

**Stylistic summaries**

In journalistic writing, it is commonplace to make the titles of articles stylistic, involving such features as wit, humour, sarcasm, rhyme, amongst others. Since the data used to train models in this project are, in fact, newspaper article-summary pairs, we find several examples of stylistics in reference summaries. Style like this is notoriously difficult for machines to learn and recognise (Chen and Soo, 2018; Ptácek et al., 2014; Reyes et al., 2012), and so we often end up with a system summary that, while totally correct and acceptable, is plain and literal. Let us look at the example below.

**Article:** Here’s a problem every network wants to have: as NBC considers which of its big batch of comedies to keep on its prime-time schedule and which to discard in favor of new entries, it is being forced to figure in the possibility — in some cases the likelihood — that some of the comedies left standing when the scheduling music stops will be quickly snatched up by a hungry competitor.

**Reference summary:** So many comedies, so many decisions.

**System summary:** For NBC it’s time for new entries.

In this example, what we see is the journalistic use of repetition in the reference summary to create a catchy title. The system summary, on the other hand, takes a more literal approach and describes what is going on in the company. The two summaries bear very little resemblance and thus the system summary will misleadingly be scored low across all metrics. Although, in this case, the system summary misses out some features of the article, this example aptly demonstrates how a literal system summary and a stylistic reference summary can cause problems for text summarisation evaluation.

**False positives**

This type of issue arises when there is very little difference — in terms of vocabulary, syntax or both — between the reference and system summaries, but the small difference there is makes a significant change in the overall meaning of the summary. One of the most common examples of this is in the case of negation,
in which just one or two words can completely change a summary’s meaning. Another example would be a change in verb tense or mood, in which a summary’s tense or mood is changed, for instance, from conditional (“something may happen”) to past (“something did happen”). While the majority of the words may match up between the reference and system summaries, leading to a relatively high score according to the ROUGE metrics, the overall sense of the summary is largely false. Let us take the below example to illustrate this.

**Article:** It’s looking more and more like the New England Patriots will pluck a defensive lineman or defensive back with their first pick in Saturday’s college draft, based on their #th-hour pursuit of Cincinnati Bengals unk rich unk, whose signing would complete the rebuilding of the offensive line, and the inevitable loss of defensive end unk unk, who received a two-year tender from the Philadelphia Eagles Monday.

**Reference summary:** Patriots sure to need defensive players.

**System summary:** Patriots pick up defensive lineman.

Here, we see an example of tense confusion: the reference summary talks of a conditional/future event, while the system summary places the same event in the past tense. The similarity between the vocabulary and syntax will lead to the system summary being highly awarded.

Let us take a more extreme example in which only one word between the reference and system summaries was different, for instance:

**Reference summary:** Patriots will definitely need defensive players.
**System summary:** Patriots do not need defensive players.

In this case, according to ROUGE’s metrics (as outlined in 5.1), the system summary will be highly awarded as it has high precision and recall compared to the reference summary. Issues related to negation like this are present in various areas of NLP (Farooq et al., 2016; Hogenboom et al., 2011; Syed et al., 2011).
8 Conclusion

This paper addresses the task of summarising noisy data from ASR output. There are several areas that could be improved if this project were to be repeated or added to. Firstly, it was noted that there was not sufficient time to fully maximise the tuning of hyperparameters. While optimisers and learning rate were tuned (see section 4.2), there were several other hyperparameters that could have been explored in order to further improve the quality of the models.

Furthermore, in section 7.1.4, we spoke of the need to train a model with a range of data, from noiseless to very noisy, in order for it to be able to summarise both clean and noisy data. This could potentially stop the issue of overfitting and allow it to be more flexible across a range of data.

It was clear from our results that there was a degree of overfitting in our models. While we were unable to determine a single reason for this, we pointed towards several reasons that may have contributed to it.

In this paper, we have shown that it is possible to create a noise-robust text summarisation system by training it on noisy article-clean summary pairs. We also explored the area of artificial error generation in order to produce noisy data on a large scale. In creating a noise-robust text summarisation model, we achieved results that were on par with those attained by a clean text summarisation system. From this point, there is much work to be done in order to improve the model, from training it on a wider range of data (both in terms of content and noise) to fine tuning hyperparameters to a greater extent. If such things are done, it will be possible to create a fully noise-robust text summarisation system that can be used to summarise all ASR output, whether clean or noisy.
Bibliography


Ptácek, Tomáš, Ivan Habernal, and Jun Hong (2014). "Sarcasm Detection on Czech and English Twitter". In: COLING.


