Addressing Challenges of Modern News Agencies via Predictive Modeling, Deep Learning, and Transfer Learning

Yaser Keneshloo

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Science and Applications

Naren Ramakrishnan, Co-chair
Chandan K. Reddy, Co-chair
B. Aditya Prakash
Daphne Yao
Eui-Hong (Sam) Han

June 10, 2019
Blacksburg, Virginia

Keywords: Popularity Prediction, Text Summarization, Transfer Learning

Addressing Challenges of Modern News Agencies via Predictive Modeling, Deep Learning, and Transfer Learning

Yaser Keneshloo

(ABSTRACT)

Today's news agencies are moving from traditional journalism, where publishing just a few news articles per day was sufficient, to modern content generation mechanisms, which create more than thousands of news pieces every day. With the growth of these modern news agencies comes the arduous task of properly handling this massive amount of data that is generated for each news article. Therefore, news agencies are constantly seeking solutions to facilitate and automate some of the tasks that have been previously done by humans. In this dissertation, we focus on some of these problems and provide solutions for two broad problems which help a news agency to not only have a wider view of the behaviour of readers around the article but also to provide an automated tools to ease the job of editors in summarizing news articles. These two disjoint problems are aiming at improving the users' reading experience by helping the content generator to monitor and focus on poorly performing content while allow them to promote the good-performing ones. We first focus on the task of popularity prediction of news articles via a combination of regression, classification, and clustering models. We next focus on the problem of generating automated text summaries for a long news article using deep learning models. The first problem aims at helping the content developer in understanding of how a news article is performing over the long run while the second problem provides automated tools for the content developers to generate summaries for each news article.

Addressing Challenges of Modern News Agencies via Predictive Modeling, Deep Learning, and Transfer Learning

Yaser Keneshloo

(GENERAL AUDIENCE ABSTRACT)

Nowadays, each person is exposed to an immense amount of information from social media, blog posts, and online news portals. Among these sources, news agencies are one of the main content providers for each person around the world. Contemporary news agencies are moving from traditional journalism to modern techniques from different angles. This is achieved either by building smart tools to track the behaviour of readers' reaction around a specific news article or providing automated tools to facilitate the editor's job in providing higher quality content to readers. These systems should not only be able to scale well with the growth of readers but also they have to be able to process ad-hoc requests, precisely since most of the policies and decisions in these agencies are taken around the result of these analytical tools. As part of this new movement towards adapting new technologies for smart journalism, we have worked on various problems with The Washington Post news agency on building tools for predicting the popularity of a news article and automated text summarization model. We develop a model that monitors each news article after its publication and provide prediction over the number of views that this article will receive within the next 24 hours. This model will help the content creator to not only promote potential viral article in the main page of the web portal or social media, but also provide intuition for editors on potential poorly performing articles so that they can edit the content of those articles for better exposure. On the other hand, current news agencies are generating more than a thousands news articles per day and generating three to four summary sentences for each of these news pieces not only become infeasible in the near future but also very expensive and time-consuming. Therefore, we also develop a separate model for automated text summarization which generates summary sentences for a news article. Our model will generate summaries by selecting the most salient sentence in the news article and paraphrase them to shorter sentences that could represent as a summary sentence for the entire document.

Dedication

To my lovely wife for standing by me throughout this journey

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my advisor Dr. Naren Ramakrishnan for his continuous support throughout my Ph.D. study and guidance that helped to shape my research. Over the course of my Ph.D., lots of bitter-sweet moments happened to me and without his unwavering support, this journey would have had a completely different ending. I am most thankful for his patience and trust in my abilities that I could achieve all that I have and more.

My co-advisor, Dr. Chandan K. Reddy for his thoughtful insights and expertise that were instrumental in helping me to improve my research at times that I was completely clueless about my next move during my studies. I have mentioned to my colleagues, countless time, that without his help and guidance, my Ph.D. could have prolonged for another year. He was always there to encourage me to refine every aspect of my research work and helped me notice the minute details which makes a good researcher even better.

My committee members, Dr. Daphne Yao, Dr. B. Aditya Prakash, and Dr. Eui-Hong (Sam) Han for guiding me throughout the process, without their valuable feedback and knowledge it would not have been possible for me to complete my Ph.D. and have a quality research defense.

My wonderful wife for her confidence in my abilities which helped me dream big. Without her mental and emotional support, this chapter of my life would have been framed completely different: dull, tedious, and bland. I am beyond lucky to have her by my side and share every moments of my life with her.

To my parents, for trusting to their first kid who was raised in a small town in Iran, to follow his biggest dreams far far away from home. I still remember the brightness in my parents' eyes after I broke the news of my acceptance to the Ph.D. program at Virginia Tech. Their immense support, both financially and emotionally, was a pivotal point for me to withstand the hardship of my studies and kept pushing myself to be a better person, everyday.

To my sister and brothers, for being buoyant at each step of the way, believing in their older brother. I am deeply humbled to have them as my family and cannot put any words for times that I missed being with them while studying abroad.

My best friends - Ahmadreza Azizi, Sirvan Paraste, Ehsan Asghari, Alireza Borhani, Mostafa Vaziri, Masoud Maghsoodi, Dr. Ali Charkhesht, and Kazem Qazanfari. Each of them, unequivocally, have played an essential role in my life. They have been there for me every step of the way, giving me their support and pushing me when I needed it the most.

I am grateful to be a part of the Discovery Analytics Center (DAC) and would like to thank all the members and alumni who have contributed to my research work and academic life at Virginia Tech. I would like to thank my esteemed colleagues: Dr. Hao Wu, Dr. Wei Wang, Dr. Yue Ning, Dr. Patrick Butler, Dr. K.S.M. Tozammel Hossain, Dr. Brian Goode, Dr. Prithwish Chakraborty, Dr. Saurav Ghosh, Dr. Parang Saraf, Dr. Rupinder Paul Khand-

pur, Sathappan Muthiah, Nikhil Muralidhar, Mohammad Raihanul Islam, Rongrong Tao, Sneha Mehta, Debanjan Datta, Subhodip Biswas, Malay Chakrabarti, Tian Shi, Ping Wang, Aman Ahuja, Khoa Doan, Ming Zhu, and Dr. Siddharth Krishnan.
I also want to take this opportunity to thank Juanita Victoria, Wanawsha Hawrami, and Joyce Newberry for their incredible support on countless occasions when I needed help.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Consuming news articles is an integral part of our daily lives and news agencies such as The Washington Post (WP) expend tremendous effort in providing high quality reading experiences for their readers. To achieve this goal, today's news agencies are changing themselves much faster from traditional journalism to modern technologies. Due to these adaptions, news agencies are able to do analysis and provide opportunities that have never been possible with traditional approaches. Although most of the current technologies used in current web-based news portals are still heavily monitored by a human-in-the-loop, such human involvements are getting more and more infrequent by the use of artificial intelligence. On the other hand, for most of these agencies, a successful technology is only adopted when it can suitably adapt itself with the dynamic of the world. Therefore, with the immense amount of content being generated everyday over the internet through social media and blog posts, the use of an automated and scalable model for analyzing such content is a necessity.

Usually a news article has different stages of progression, starting from the editor's write up following up by its publication on the web and finally modifying the content according to the feedback received from users to maximize the quality of reader experience. Therefore, it is not surprising that news agencies and specially news editors will think about the performance of a news article not only after its publication but also before starting to write the content. To help the editors and news agencies at each of these stages, they utilize various methods in data mining, machine learning, and artificial intelligence. Data mining provides various opportunities for an editor or the content manager to have a broader view of the performance of a news article from different angles even before its publication. In general, news agencies will monitor the performance of a news content via the number of clicks that it receives over a specific period of time (e.g. 24 hours) and depending on how much attention a news article receives, they will modify the content. For instance, if a news article is marked as viral, they can promote it on various social media or move the news to the center of attention in the web portal while for a news article that is not receiving as many viewers, they might change the title, summary, or content of the news to boost the performance of it. Therefore, one of the challenges for these news agencies is to have a proper and accurate analysis over the performance of each news article and have various tools to help and facilitate the improvement of each news content.

On the other hand, machine learning and artificial intelligence provide tools that could help the news agencies to facilitate some of the works, that has been done traditionally by human,

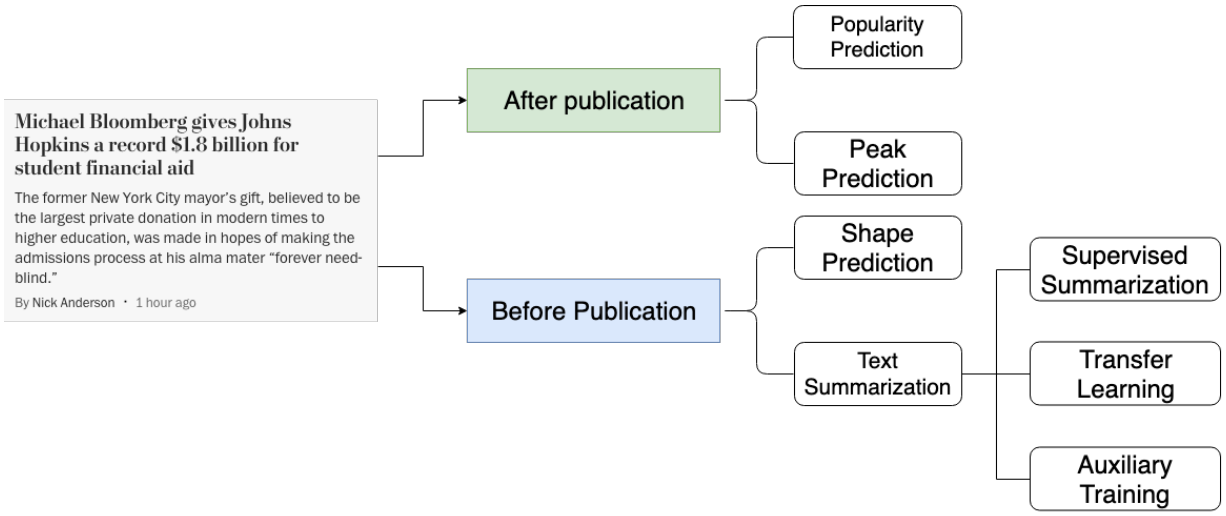


Figure 1.1: The overall research problems that we solve throughout the thesis.

much more efficiently by a machine. Specifically, having publishing more than thousands of news articles each day will create challenges such as selecting the best title and summary for each news article such that it will maximize the number of viewers for a news agency. Due to this massive amount of data, using human-written summary is not only infeasible but also very time-consuming. Luckily, with recent progresses in deep learning, various automated summarization models are proposed for news articles which either generate a news headline by shortening the first sentence of a news article [101] or provide a three to four sentences summary after reading the entire document [21, 105]. Although there has been a lot of researches on how to create eye-catching titles that play like click-baits for users [67], professional news agencies, in general, avoid using these practices and focus on generating higher quality headlines and summaries.

1.1 Research Problems

In this dissertation, we explore various models that help a news agency to both analyze the performance of the published content and facilitate the some of the works that was previously done only by a human through.

Fig 1.1 connects these problems and categorizes them into two different categories:

- After Publication: After publication of a news article we are interested in figuring out how a news article evolve over time by predicting its number click-views and peak view time.
- Before Publication: Before publication of a news article, we will first predict the shape

of a the click-view of a news article which later on will be used by the peak prediction model. On the other hand, we will focus on the problem of news text summarization from various angles.

1.1.1 Popularity Prediction of News Articles

Journalists and editors are faced with the task of determining which articles will become popular so that they can efficiently allocate resources to support a better reading experience. News agencies such as The Washington Post (WP) publish more than a thousand pieces of news content every day. However, not all of these articles become equally popular, and thus popularity prediction is an invaluable strategy for journalists and editors to prioritize which articles need to be refined. The popularity of news articles is typically dependent on various elements, involving contemporariness, writing quality, and other latent factors. However, identifying the right factors involved in performance of a news article requires deeper analysis on understanding the effect of each of these factors.

Therefore, two issues are critical in the resolution of the popularity prediction problem:

- **How to measure popularity of a news article**: People consume their news via a variety of channels nowadays which varies from mobile devices, social media, blog posts and applications that are designed for personalizing news articles. Therefore, there are multiple measures of popularity, e.g., number of page views on the news agency site, number of likes or shares on social media, or the number of searches in a search engine. Choosing which one of these measures is the right medium for evaluating the popularity of a news article is hard and infeasible since it requires accessing to a broad and huge amount of data. Moreover, due the variety of mediums that people use to consume these news article, there might not necessarily exist any correlation between the data that we see from one channel to the other. For instance, most of the people reads news article through the social medias like Facebook, however they may never click on the link provided on the post to be directed to the main news agency portal. Therefore, to reach a broader view of popularity of a news article, we will require a vast amount of data. Although with all these difficulties that exist in understanding of the true popularity of a news article, measures such as the number of page views on the news agency web portal provides the one of best and accurate ways for evaluating the popularity of a news article.

- **Local vs. Global popularity**: Local context measures are primarily meant for use within a single news agency whereas global context measures help ascertain the popularity of an article amongst articles from other new agencies, as well. Similar to the first problem, while our ultimate goal is to integrate a range of popularity measurements across different news channels in a global context, due to infeasiblity of collecting data among these channels, we need to focus on local context. we primarily

focus on predicting popularity in a local context and use the number of page views of an article as a surrogate for its popularity.

Once we settle on the right measure that captures the popularity a news article as accurately as possible, we will focus on methods that could be used to provide prediction over this measure. We will use a combination of various features collected either from the news article content or its metadata along with temporal features collected from the click time-series of the news article, to provide an accurate prediction model that is able to provide estimation of the number of clicks that a news article receives after 24 hours by just looking at the data that are collected within the first 30 minutes after its publication.

After solving the popularity prediction of a news article, we will shift our focus to a similar but deeper analysis on understanding how a news article evolves over time. The evolution of a news article could be understood by either looking at the pattern of the click time-series over time or via predicting the time that a news article reaches its maximum number of viewers. Therefore, we propose the following two problems which deals with understanding how a news article evolves over time:

- **Predicting the click pattern of a news article**: For this problem, we are interested in understanding how a news article will evolve over time after its publication. Questions like "is this article going to have a viral behavior?" or "will it have a slow-paced readership over time?", are the two main questions that are being answered through solving this problem. We provide a solution for this problem, by predicting the pattern of a news article with information that are collected before the publication of a news. This will provide a good analytical tools for the editors and content managers to modify their news article according to the evolution pattern that is expected for them.
- **Predicting the peak time of a news article**: Although predicting the evolution pattern of a news article will provide a general idea on how the readership of the news article evolves over time, it does not provide detailed analysis over this evolution. Among all different time points in the lifetime of a news article, news agencies are interested in predicting the peak time that a news article reaches its maximum number of viewers. This could benefit the news agencies in a better targeted advertisement and even enriching the content of a news article during the vicinity of the peak time for maximizing the readership.

1.1.2 Actor-Critic Model for Text Summarization

In modern news portals, each news article is accompanied with two important factors that draws attention of readers: A news headline and its summary. The headline of the article is usually a very short sentence that plays a crucial role for a user to identify whether a

news article is of his/her interest. The second most important factor for reading a news article is the summary sentences provided by the news agency for a specific news article. These summaries are usually comprised of two to four sentences which will provide a brief overview of the content of the entire news article. Generally, with the massive amount of data that are exposed to each person, we will be more selective on the contents that we are willing to consume and only click on a news article if the title and summary of the article are of our interest. Therefore, putting focus on providing the best title and summary for each news article is of importance for these news agencies. However, generating human-written headlines and summaries for thousands of news article on a daily basis is not only infeasible but also time-consuming and expensive. Thus, news agencies are interested in automated tools that provide the best headline and summary for a news article.

Traditional models in text summarization mostly focused on compression-based and language specific models [57, 123] and perform poorly on long documents, e.g., a news article. However, with recent advances in deep learning, we are able to create models that could read a long document and provide short summaries for it. Although these models are suitable for generating both the headline and summary of news article, we focus on the summarization task and provide solutions for this problem using a combination of various architectures in deep neural networks.

Text summarization is broadly divided in two different categories:

- **Extractive Summarization**: In extractive summarization, the goal is to select the most salient sentences in a document and present it as the summary of the whole document. However, usually sentences in the original document are very long and contain extra information that are not suitable for a summary.
- **Abstractive Summarization**: On the other hand, in abstractive summarization much like human summaries, the goal is to generate a completely novel summary sentence after reading the entire documents.

Recent text summarization models are mostly focused on one of these two problems. However, a suitable text summarization model requires to have both of these characteristics. It not only needs to be able to correctly select the most salient sentences, but also it should be able to generate novel words that are not present in the document. On the other hand, since these two models are disjoint component of a summarization framework, we will use a special technique in RL known as Actor-Critic learning to combine these two modules and provide a differentiable uniform framework for this problem.

One of the known characteristics of training of deep neural models is that they are data-hungry models. This means that these models require a large dataset that should contain ground-truth human-written summary for each news article. Currently, there are two datasets that provide such large data known as the CNN/Daily Mail dataset [40] and Newsroom [37] dataset. Each article in these datasets is accompanied by one to four human-

written summaries. Similar to most of the recent works in this area, we will also use these datasets to build our text summarization model.

1.1.3 Transfer Learning in News Article Summarization

Although current state-of-the-art (SOTA) models, could provide a good abstractive summarization on datasets such as CNN/Daily Mail (CNN/DM) dataset, they will lack to generalize on other datasets. According to our preliminary analysis, we observed that current SOTA models suffer from generalizing to other datasets. If we train a text summarization model on news articles that exist in CNN/DM dataset, the performance of the trained model will be poor on news article that are coming from The Washington Post dataset. This observation led us to the idea of using transfer learning to improve the result of text summarization model on unseen datasets. Current techniques in transfer learning are mostly focused on image applications [26, 131] and there are very limited works that used these methods for Natural Language Processing (NLP) tasks [106]. In transfer learning, we will usually pre-train a model on a specific dataset and then fine-tune it on the target dataset. In transfer learning and during the pre-training stage, the model requires a huge amount of data but during fine-tuning, we can only use a small number of samples to change the distribution of the model to get better results on the target dataset.

We are interested in creating a framework that uses a large datasets such as CNN/DM to pre-train a summarization model and later on uses a smaller dataset to fine-tune the trained model such that we receive better results on the target dataset.

1.1.4 Text Summarization with Auxiliary Training

One of the benefits of having a text summarization model is the ability to use the summary as a surrogate for the whole article. This surrogate could be potentially used in a lot of downstream tasks such as text classification. Therefore, rather than using the whole article (which contains a lot of irrelevant text for the underlying task and is usually very long), we can use the summary of the document to do the analysis. To achieve this ability, one might think to combine the summarization and the underlying task so that the summarization algorithm tries to generate sentences that not only better capture the content of the entire document, but also could improve the performance of the downstream task, e.g. text classification.

A solution for these types of problem usually fall into the multi-task and auxiliary task learning. In multi-task/auxiliary-task learning, we will try to improve not only one but many sub-problems using one unified framework. In multi-task learning, we usually want to achieve a better performance for all the underlying tasks while in the auxiliary-task learning, we only have one primary task and we use all the other tasks to improve the performance of

this primary task and do not care about the performance of auxiliary problems.

Our main objective is to achieve a better results in text summarization. Thus, we will use various auxiliary tasks relevant to the summarization problem, to potentially improve the performance of the text summarization models.

1.2 Organization of the Dissertation

As mentioned before, we will study various models that could help a news agency to provide better management over the contents they want to present to the users. To this end, we will divide our studies to two different challenges where the first challenge focuses on understanding how a news article evolves after its publication while during the second problem, we will automate the text summarization problem which was traditionally done by human. The solution to the first challenge will give a broad overview of the performance of a news article in the long run while the second one will reduce a lot man power required for summarization of news articles and thus saves massively both in time and money. Given these challenges, the organization and goals of this dissertation are as follows:

In Chapter 2, we discuss about the popularity prediction problem and present some of the previous works that are have tackled this problem on similar content. We then propose our model which solves this problem from three different angles as mentioned in Section 1.1.1. We first present our work on predicting the popularity of a news article after 24 hours using both temporal and contextual features that we collect after 30 minutes of publication of a news. Then, we move on with with the problem of predicting how the news article readership evolves over time and then expand it to provide an estimation on the peak time of a news article.

In Chapter 3, we shift our focus to the text summarization problem and propose a new extractive-abstractive text summarization framework which uses reinforcement learning to achieve successful and state-of-the-art (SOTA) result on a variety of datasets. Although our proposed method in Chapter 3 will provide the SOTA result on various datasets, it still lacks to generalize to other datasets. This problem is a well-known problem in transfer learning and we will propose some ideas on how we can tackle this problem. Therefore, in Chapter 4 we will provide some of the preliminary analysis that prove how current SOTA models in text summarization suffer from the generalization to other dataset and propose a solution based on reinforcement learning for this problem.

To better improve the result of text summarization model on various datasets, we augment this problem with various other similar tasks, such as sentiment analysis using auxiliary learning. Chapter 5 will provide the solution for this augmentation and how we avoid some of the major problems in the auxiliary training with multiple disjoint tasks. Finally, we conclude the dissertation in Chapter 6 and provide some insights on the potential future works on these problems.

Chapter 2

Popularity Prediction of News Article

News is an integral part of our daily lives and agencies such as The Washington Post (WP) publish more than a thousand pieces of news content every day. However, not all of these articles become equally popular, and thus popularity prediction is an invaluable strategy for journalists and editors to prioritize which articles need to be refined. In this chapter, we will study the popularity prediction of a news article from two different angles:

- Predicting the number of views of an article after 24 hours using the data collected within the first 30 minutes of its publication.

- Predicting how a news article evolve over time and when it reaches its maximum number of readers.

For the first problem, we use contextual information collected before the publication of the article and use temporal features collected 30 minutes after its publication. While for the second problem, we use contextual information that are published before the publication of article to predict the evolution pattern while we use temporal information collected after its publication for predicting the peak time.

2.1 Washington Post Dataset

The Washington Post granted us access to a dataset of news articles that are clicked at least once during the period of September 1, 2014 until October 31, 2014. For each news article that are published during this period, we have access to the metadata information assigned for each article along with the click time-series with second granularity in time. We then aggregate these click time-series to five-minute intervals and use them during our analysis. Overall, this dataset contains more than 41K articles that are published between September and October 2014, and out of which metadata information is available for 37K articles. Therefore, for about 4K articles in our dataset, we have missing metadata features for whom we use zero as the default value.

2.2 Predicting 24 hours Click-rate

Two issues are critical in the resolution of the popularity prediction problem. First, since people consume their news via a variety of channels nowadays, there are multiple measures of popularity, e.g., number of page views on the WP site, number of likes or shares on Facebook, or the number of searches in a search engine. Second, article popularity can be defined in a local or a global context. Local context measures are primarily meant for use within a single news agency whereas global context measures help ascertain the popularity of an article amongst articles from other new agencies as well. While our ultimate goal is to integrate a range of popularity measurements across different news channels in a global context, we primarily focus on predicting popularity in a local context and use the number of page views of an article as a surrogate for its popularity.

By its nature, the life span of a news article is very short after its publication. Interviews with journalists and editors at The Washington Post suggested that it is more interesting and valuable to predict the early popularity of an article rather than its long-term popularity. Therefore, in our study, popularity of an article is defined as the number of page views within the first 24 hours following publication. We cast popularity prediction as a regression problem, extract features from a news article for up to 30 minutes following publication, and use these features to project the page views the article will receive within 24 hours.

Our main contributions for this study could be summarized as follows:

1. We evaluate an extensive set of metadata, contextual or content-based, temporal, and social features to predict the popularity of a news article. For instance, in addition to the click-stream and full content of the articles, we utilize features from Twitter users who shared the articles, features estimating freshness of an article, as well as sentiment features.

2. We evaluate multiple regression models for popularity prediction and deploy our best performing models in a real-time system at The Washington Post.

2.2.1 Related Work

Popularity prediction for news articles is a relatively novel problem and very few studies addressed this problem. However, a growing number of studies have been carried out on predicting the popularity of other types of online content. Several studies have analyzed the rate at which tweets diffuse on Twitter or that at which videos are viewed on YouTube. The goals of these studies include predicting the exact number of retweets for a tweet [133], predicting the number of YouTube views for a video [38, 110], estimating the number of votes to a Digg post [56] or the number of page views for a news article [69], ranking news articles [112], forecasting the ranges of popularity for a tweet/news article [7], and prediction of the exact number of comments for a news article [112, 115].

The prediction of online content popularity can be undertaken at two stages: before or after content publication. There are many studies that focus on using the early measurements after publication to predict future success [69, 110]. On the other hand, studies such as [7, 115] aim to predict the popularity before publication. Bandari *et al.* [7] used the number of times an article is posted on Twitter along with some contextual features to predict tweet counts. Mentions in tweets can be used as a surrogate for popularity to some extent but this is less accurate than page views on an article (as used by our proposed method). In another study, Zhao *et al.* [137] used the number of followers of the user who retweets a post to predict the total number of retweets for a tweet.

A Bayesian approach is proposed by Zaman *et al.* [133] to predict the number of retweets of a tweet according to two features: number of followers of retweeters and the depth of the retweeter in the retweet tree. Using a graphical model, this approach trains different parameters related to these features. It uses the reaction time of a user, i.e. the time between when user sees a post and when the user retweets it, as the main predictor to predict the final retweet count.

Existing studies frame the popularity prediction problem as either regression [54], classification [115], or even clustering [38]. A variety of features are used in these studies to predict the popularity of tweets/news articles [113]. We can categorize these features into: content-based and temporal features.

Content-based features are usually extracted from the text of a tweet/news article. Features such as the sentiment of a text [11, 96], emotions within the text [12, 13], subjectivity of its language [7], named entities [7], and freshness of a content [16, 20] are all considered as highly correlated factors to virality of content. Bandari *et al.* [7] suggested the idea of using the category and the name of the website that publishes the article to predict virality. Fig 2.1 plots the distribution of categories for the viewed and tweeted articles in our WP dataset. As can be seen, categories and their corresponding distributions differ between the articles posted on Twitter and the ones viewed on the WP website. This shows that users do not necessarily share the content that they read with their friends. Instead, they select specific stories and share them among their network. This result follows the finding in [14], where they found that users only tend to share selected stories with their friends.

Temporal features are mainly extracted from the click time-series of an article. Among all the temporal features, the number of retweets/clicks in the first hour of publishing the tweet/article (n_0) is known to be highly correlated to the final counts of retweets/clicks at time t, i.e., n_t [38, 110]. According to Szabo *et al.* [110], there is a high linear correlation between the log-transformed number of retweets/clicks in the first hour and its long-term popularity. This study proposes a simple constant multiplier α to estimate n_t according to n_0, i.e., $\log n_t = \alpha \cdot \log n_0$. We use a similar approach as one of our baseline models. An extension of this method is proposed by Pinto *et al.* [92], where they replace n_0 with samples at regular intervals (15 minutes) up to the first hour. However, the relative importance of the clicks of an article, among all the other articles that are published at the same time,

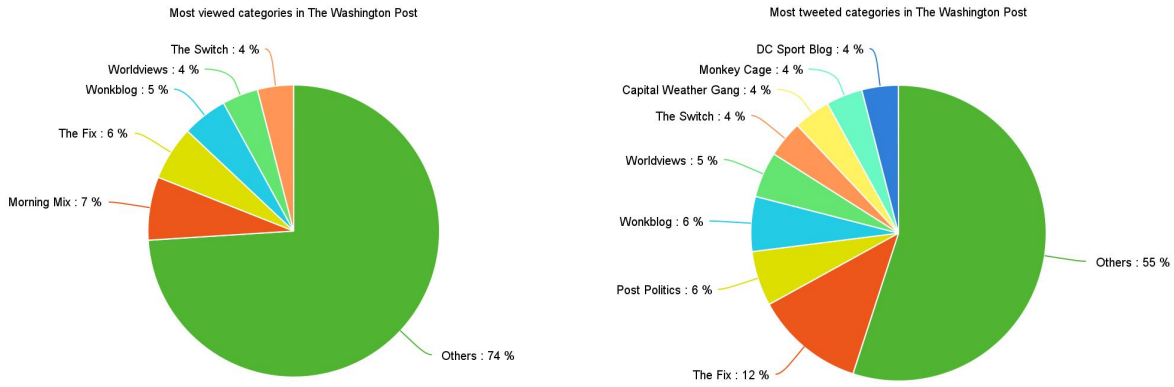


Figure 2.1: Distributions of categories for (left) most viewed and (right) most tweeted articles in The Washington Post website. Note that the distributions differ.

is ignored in this method. We improve this feature by suggesting a normalized page view feature which is relative to all the articles that are published at the same time. Along with these features, retweet acceleration and the retweet depth in the retweet tree have also been used to predict the popularity of Twitter messages.

2.2.2 Proposed Method

In this section, we present our proposed method to predict the popularity of a news article. Our goal is to predict the number of page views that a news article will receive within the first day after its publication. We track all articles for 30 minutes upon publications and extract a range of temporal, social, and contextual features for forecasting. In this section, we briefly explain each of these features and talk about how each of them contribute to the popularity prediction problem.

Metadata features

The WP dataset contains detailed information about each article such as title, full content, keywords, authors, type (blog or article), category, news section, and the publication date. From the publication date of an article, we extract the hour of the day and day of week that the article is published. Along with these features, we use the author name, news type, category, and section as our metadata features.

Content-based features

Contextual features deal mostly with the title, keywords, and the content of the article. We separate these contextual features into two sets: the first group includes features extracted from the WP metadata dataset and the second group is extracted by querying our *pseudo* archive of the WP dataset (explained in detail later).

1. **Sentiment.** As our first contextual feature, we extract the sentiment of a text segment as probabilities belonging to either positive, negative, neutral, or compound classes. We use the Vader sentiment analyzer [43] for this purpose. In addition to the sentiment of a text, the emotion of a text is also an important factor in influencing virality [12]. However, as discussed by Berger and Milkman [12], there is no linguistic tool that can capture emotions in a text, and we have to determine the emotions in the text, manually. We create two different sentiment feature sets, one for the sentiment of the full content and the other for the titles of the news articles. We use these two different sets because most of the current sentiment analyzers face a problem when classifying long text. Each sentiment contains a probability that shows the likelihood of the content being positive, negative, neutral, or compound. In our WP dataset, there is a 0.42 correlation between the positive sentiment probability of content and title, 0.23 correlation among the negative sentiment, 0.18 for the neutral sentiment, and 0.34 for the compound sentiment. As you can see, the correlation is not significant for any of the sentiments. Therefore, we use both the title and content of an article to capture the sentiment.

2. **Named Entity Extraction.** Named entities are also an important factor in influencing the virality of news articles. News articles about a well-known location or a person can bring a lot of attention to itself. We divide these named entities to three different categories comprising of locations, organizations, and persons. For each category, we count the number of named entities that exist in the article for this category. We use the Stanford NLP Library [1] for this purpose. Although capturing only the number of named entities may not capture the importance of each of these entities, it will provide an indicator of how thoroughly the article talks about a subject. In our experiments, we will show how these numbers help to improve our predictions. As described later, we also capture the importance of named entities by estimating counts from the *pseudo* dataset.

3. **Readability.** Berger and Milkman [12] claimed that longer articles tend to be shared far more often. On the contrary, we found that the correlation between the length of an article and the number of page views that it receives is small, the correlation value in our WP dataset was 0.08. The correlation that is found in [12] is not a causation, and it is possible that some other factors cause this correlation. However, this claim

[1] http://nlp.stanford.edu/software/

is based on the premise that journalists tend to write longer pieces when they are writing on hot topics. Along with the article length, the readability of the article is also a factor and there are several different metrics for this purpose (typically based on estimating the number of years of education required for a person to understand the text). Most of these methods use a combination of word and sentence length, number of complex words, and number of syllables within the text to estimate readability. For our purpose, we use the Flesch-Kincaid Readability Test, Gunning-Fog Score, Automated Readability Index, and the Coleman-Liau Index to determine readability [2]. Along with these metrics, we also consider the number of sentences, number of complex words, percentage of the complex words to the total number of words, number of syllables, average number of syllables per word, average number of words per sentence, average sentence length, and length of the title and full content of an article.

4. **Freshness.** All aforementioned features are measured using standard methods and software libraries. In this section, we propose a method that aims to determine the freshness of an article. As mentioned earlier, viral news articles are usually driven by fresh and surprising information. In order to capture the freshness of an article, we must model articles coming from other news agencies in the recent past. However, accessing to such broad information is infeasible. Therefore, as a solution to this problem, we create a *pseudo* historical dataset containing WP articles published in the past (before Sep 2014) that received at least one page view in the period of Sep 2014 to March 2015. We then use this dataset as a surrogate to find similar news articles. The pseudo-archive is indexed using Lucene [3] w.r.t. the title, full content, and keywords and we use Lucene's built-in scoring mechanism to identify the top-10 most similar articles. From the similarity results, we extract the following features that could indicate the freshness of an article:

   - **Topic intersection.** The topic intersection between the queried article and the archived articles is defined as:

$$TI = \frac{|\text{keywords}_\text{q} \cap \text{keywords}_\text{a}|}{|\text{keywords}_\text{q} \cup \text{keywords}_\text{a}|} \tag{2.1}$$

     where keywords_q and keywords_a are the sets of keywords in the queried article and the top ten articles, respectively. This feature will be close to zero for fresh or new articles and close to one for old stories.

   - **Top ten stories page view count.** For each of the top ten similar articles, we find the number of page views that the article received until the publishing date of the queried article. We consider these features as TC_i, for $i = 1, \cdots, 10$ sorted so that $TC_i < TC_{i+1}$. These features will provide us a rough estimate of the virality of the new article w.r.t. the performance of historical similar articles.

[2] We refer readers to read the paper by Dubay *et al.* [30] for a description of these measures.
[3] http://lucene.apache.org/

- **Top ten stories content similarity.** For each of the top ten articles, we find the cosine similarity of its content against the content of the queried article. This feature also captures the freshness of content. We represent these features as TS_i, for $i = 1, \cdots, 10$ sorted so that $TS_i < TS_{i+1}$.

Along with the above features, we also capture the total number of articles that are similar to the queried article. We call this feature the *hits* number, which represents the number of hits for the queried article in our archived database. The lower this value, the more fresh the content of an article.

Temporal features

We extract temporal features from the click stream data of page views at 5 minutes intervals. In analyzing YouTube views, it has been found that there exists a high correlation between the log-transformed number of views in the first hour and the ultimate number of views an entry receives [110]. While other works, such as [50], used additional features such as the retweet time series in the first 30 minutes and retweet acceleration, in our proposed method, the number of page views in the first 30 minutes is considered as the primary feature for prediction. Fig 2.2 illustrates the correlation of the page views in the first 30 minutes with their 24 hours page views in the WP dataset. We find that while a small correlation exist between the two measures, the observations are scattered more in space, suggesting that such an estimation can at best be used as a bound. Here, we propose additional features in conjunction with this feature to predict the number of page views:

- **Time difference between the publishing time and first page view.** This feature captures how fast people react to a news article. We also call this feature *page view reaction time*. We expect that for viral articles, this number will be small and for ordinary articles this be a higher number. In our WP dataset, the median time difference for all articles is 237 minutes while 85% of the top 1% viral articles have less than 200 minutes reaction time.

- **Number of page views after 30 minutes.** This number captures the total number of page views that an article receives within the first 30 minutes after its publication. On average, the top 1% articles received around 571 page views after 30 minutes and 300K after a day. It is worth mentioning that the average page views for the first 30 minutes of the rest of the articles is around 18.

- **Page view acceleration.** We use the approach from Kong *et al.* [50] to capture page view acceleration as follows:

$$\text{Acceleration} = \frac{\sum_{t=2}^{N} n_t^x - n_{t-1}^x}{N} \tag{2.2}$$

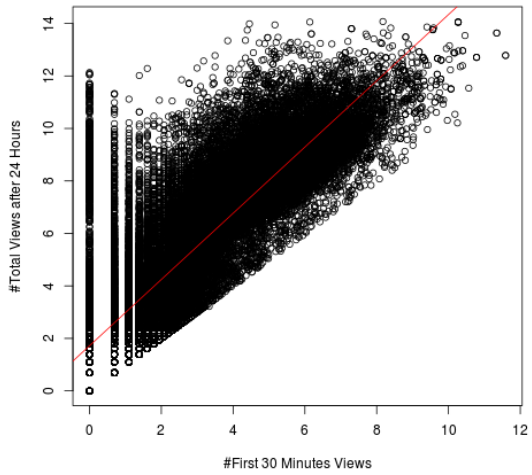


Figure 2.2: Correlation between the log-transformed number of first 30-minutes page views and the total number of page views after 24 hours for the WP articles. The red line represents the simple linear regression estimation for the first 30 minutes page view count. The significant amount of scatters around this estimate suggests that additional features are necessary for improving performance of the prediction.

where n_t^x is the number of page views of article x at time interval t and N is the total number of time intervals within the first 30 minutes. Since we use a 5 minute time interval to build our time series, $N = 6$ for the first 30 minutes.

- **Page view time series.** Similar to Kong *et al.* [50], we also use the values in each time interval of page view time-series, i.e. n_t, as predictors.

- **Normalized page view time series.** We normalized the page view time series w.r.t. the time series of other articles published at the same time. Therefore, given m articles that are published at the same time, the normalized time-series is as follows:

$$NC_t = \frac{n_t^x}{\sum_{i=1}^{m} n_t^i} \tag{2.3}$$

where n_t^x is the total number of page views of article x within the time interval t and n_t^i is the total number of page views of i^{th} article that is published at the same time as article x. To collect these values, we use a time window to normalize these time series w.r.t. all articles published in the last few hours. The normalized count is a better measure than the count itself, since it finds the relative importance of an article among other articles that are published at the same time. Given this score and having m articles in a specific time-frame, the one that gets more attention will have a higher normalized count. Therefore, a normalized count close to one means that the article is receiving more attention amongst all other articles published at the same time.

Social Media Features

Using the click stream dataset, we generate another dataset to capture social media (Twitter) activity related to each article. We use the Twitter API[4] to extract all tweets that share a WP URL. For each article, we create a tweet and retweet time-series similar to the click stream dataset, i.e. each row of this dataset contains the timestamps of the tweet along with its TweetID. Using the TweetID of each tweet, we query Twitter using its API to access the user profile and generate a user profile dataset. For each user, we extract the number of followers, number of listed counts, number of friends, and number of status message/updates. For each news article, from the extracted tweets related to each article, we build the retweet time series. We use the retweet time series of an article to generate another set of social media-driven temporal features:

- **Time difference between publishing time and first tweet.** We call this feature the *tweet reaction time* and like the *page view reaction time*, the tweet reaction time captures how fast people share the news on Twitter. Similar to the *page view reaction time*, we expect that viral articles are shared faster than other news articles. In our WP dataset, the mean and median of the tweet reaction time, for articles that have at least one tweet in Twitter, is 13 minutes.

- **Number of retweets after 30 minutes.** This number finds the total number of retweets that an article receives in 30 minutes following publication. On average, the top 1% articles received around 106 retweets after 30 minutes. Surprisingly, the rest of articles receive around 194 retweets on average after 30 minutes. The reason for this behavior is that, as mentioned in Kwak *et al.* [53], people use social media to share almost everything that happens around them. However, only a small portion of these news articles go viral. We will later see in Section 2.2.3 that unlike the page views in the first 30 minutes, the retweet counts do not significantly contribute to prediction performance.

- **Number of '30 minutes followers'.** We extract the number of followers of all users who shared the article and aggregate them to find this number. This feature captures the approximate number of people who were exposed to the article within the first 30 minutes. On average, the aggregate number of followers of users who shared the top 1% articles is around 380, while for the rest of the articles this number is 14. This shows that on average people who share viral articles tend to have more followers.

- **Retweet/followers acceleration**: Similar to page view acceleration, this feature is calculated using an equation akin to Eq. (2.2).

- **Retweet/followers time-series**: We use the values in each time interval of retweet-/follower time-series, i.e., n_t, as predictors.

[4]https://dev.twitter.com/

Table 2.1: Features evaluated in this work.

Metadata Features	
Article Type	Whether the article is a blog post or article.
Article Category	Different categories are viewed by different people; some categories do not not usually generate viral articles; others generate more popular news.
Article Section	Similar to the category of an article, the section of an article is also an important factor in influencing its popularity. Most published articles fall in the Sports, Politics, and Opinion sections.
Publication Date	We record this feature in terms of the time of day and day of week that the article is published.
Author Name	There are more than 12k authors in our dataset; authors such as Valerie Strauss and Dan Steinberg publish the most articles.
Contextual Features	
Sentiment	We extract sentiment scores for both the title and the main article. The sentiment is defined as probabilities in four categories: {negative, positive, compound, neutral}.
Named Entities	Number of persons, locations, and organizations in an article
Readability of Text	Five measures that captures different aspects of readability of a document
Freshness of Article	1- Topic Intersection 2- Click count of 10 most similar articles and its average 3- Content similarity of 10 most similar articles and its average 4- Number of similar articles in the historical dataset 5- Maximum similarity and maximum click counts 6- Number of viral articles in the top 10 most similar articles 7- Is the article with maximum similarity a viral article? 8- Is the article with maximum count a viral article? 9- Burst time and burst time slot of the article with maximum similarity 10- Burst time and burst time slot of the article with maximum count
Temporal Features	
FirstViewTimeDiff	Time difference between publishing time and first page view.
First 30 Minute View	Number of page views after 30 minutes of publication.
Page View Acceleration	The rate at which an article is read within the first 30 minutes.
Page View Time Series *Normalized Page View Time series*	Time series of views in the first 30 minutes organized in 5-minute intervals
Social Features	
FirstTweetTimeDiff	Time difference between the publishing time and first tweet.
First 30 Minute Tweet Volume	Number of tweets after 30 minutes of publication.
First 30 Minute Followers' Number	Number of followers of users who post the news within 30 minutes.
Tweet/Follower Acceleration	The rate at which an article is being tweeted within the first 30 minutes.
Tweet/Follower Time Series *Normalized Tweet/Follower Time Series*	Time series of tweets in the first 30 minutes organized in 5-minute intervals.

- **Normalized retweet/follower time-series**: This feature is also calculated similar to how we calculated the normalized page view time-series.

Table 2.1 summarizes all the features used in our study.

2.2.3 Experiments

In overall, we extract 105 features for each article. These features are organized in sets and evaluated for their incremental improvement over the baseline method described earlier. All experiments are conducted using 10-fold cross validations and we use the average Adjusted R^2

Table 2.2: Comparison of different regression models on the complete (test) dataset and the top 1% (viral) dataset.

Model	Complete ($AdjR^2$)	Top 1% (R^2)
Multi Linear Regression	79.4	**78.2**
LASSO Regression	72	52.1
Ridge Regression	80.3	54.5
Tree Regression	**82.9**	42.5

($AdjR^2$) value to report the performance. Besides overall performance, we are also interested in the performance of the models on viral articles. Therefore, in each fold, we first train the models on the training set and then test it on two test sets: one containing all test articles in that fold for the run, and the other containing only 1% of most popular articles in the first test set.

Model Selection

Table 2.2 compares the performance of multiple regression models over our datasets. As shown here, tree-based regression performs the best on the complete dataset but performs poorly on the viral dataset. On the other hand, the multiple linear regression (MLR) model has satisfactory performance over both test datasets, and we utilize this approach for the rest of our experiments.

The baseline method utilizes the strongest signal for predicting page views, i.e., the log-transformed number of page views in the first 30 minutes after the publication of article. Our dataset contains more than 41K articles that are published between September and October 2014, out of which metadata information is available for 37K articles. Therefore, for about 4K articles in our dataset, we have missing metadata features for whom we use zero as the default value. Additionally, for articles that have not been tweeted, we use custom default values to fill the missing features. For instance, for the time difference between the first tweet and publication date of an article, we use an extremely large value for articles that do not have this measurement. For other social features such as the number of retweets/followers in the 30 minutes and the retweets/followers time-series, we use zero. The following results show that our model is robust enough to deal with missing values.

Table 2.3 shows the result of our regression analysis on this dataset. The bold entries in this table show the feature set that provides the best boost w.r.t. the baseline model. Adding only metadata features provide the maximum boost among other set of features. If we use the full set of features, we improve the performance of the baseline method by 10%. In order to show that the small improvement achieved using temporal features is also a

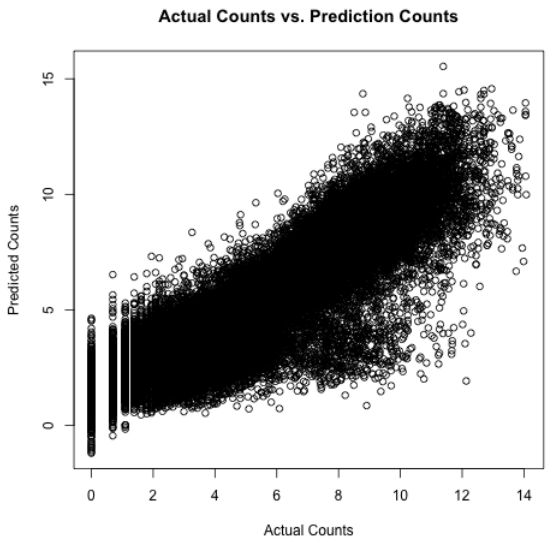


Figure 2.3: Plot of actual page views versus predicted values using our complete feature set.

significant improvement, we use the paired t-test to examine the significance of the results, and find that the boost achieved using the temporal feature is extremely significant with a p-value < 0.0001. Additionally, to better understand the effect of freshness features in the performance of the model, we remove these features from the content features. Although the absence of the freshness features causes only a slight deterioration in performance, this difference is extremely significant according to the t-test.

To see the performance of our model on the most viral articles, we use our trained model to predict the page views over this set. For this experiment, as the number of features exceeds the number of samples, we use the R^2 measure to report performance. Table 2.3 shows the performance of our model on this dataset. The baseline R^2 for this dataset is 69.4, and using the full feature set provides a 4% boost over the baseline. According to Table 2.3, although metadata information provides the most lift in the $AdjR^2$ score, the content-based features help the most in predicting the viral articles. Fig 2.3 shows the plot of actual click counts after 24 hours versus the predicted value of our whole model in the complete dataset. By comparing this plot to Fig. 2.2, we see that our proposed model will provide a much less scattered prediction and our prediction lies more accurately on a straight line.

Important Features

In order to quantify the important features in our dataset, we explore each set of features separately to identify sets that provide the maximum boost in model performance. We aim to extract the best subset of features that provide the maximum $AdjR^2$ value. Tables 2.4 and 2.5 depict the result of this experiment on the complete dataset and the top 1% dataset,

Table 2.3: The Adjusted R^2 and R^2 value for the regression model on the complete and top 1% datasets.

Model	Complete ($AdjR^2$)	Top 1% (R^2)
Baseline	69.4	74
Baseline + Temporal	70.4	72.1
Baseline + Social	72.5	77.3
Baseline + Content	71.1	**79.3**
Baseline + Content - Freshness	70.6	79
Baseline + Metadata	**77.2**	78.1
All Features	79.4	78.2

respectively. According to this experiment, among the temporal features, the page view time-series and its normalized time-series are the most important features in both datasets. Note that when we use only these two features to predict the page views in the top 1% dataset, we receive a higher R^2 value than when we use the full set of temporal features. This is due to possible multi-colinearity that exists between some of the features in the temporal feature set.

Similarly, the important social features are identical between the complete dataset and top 1% dataset. From all the social features, the time difference between the first tweet and the publication date of the article, number of followers of users who shared the article after 30 minutes of the publication, and the retweet time-series and its normalized time-series are the most important features. As can be seen, the performance using these selected features is exactly the same as the results in Table 2.3.

Although, as shown in Table 2.4 and 2.5, the complete and top 1% dataset share similar social and temporal features, they have different set of important contextual features. According to our experiments, in the complete dataset, a subset of top ten stories page views and content similarity was picked by the subset selection filter. Moreover, out of all the sentiment features, only the probability of content neutrality and compoundness is considered to be important in the prediction. Also, from the four different readability measures, the SMOGIndex was picked by the best subset selection method. Additionally, despite the known claim that article length is a good attribute to predict popularity [12], according to our analysis, title length is the feature that helps improve performance.

In the top 1% dataset, however, the topic intersection is one of the important features. Moreover, the compoundness of the content is not an important feature and none of the readability features are selected for this dataset. As can be seen in Table 2.5, among all the contextual features, all the proposed features related to the freshness of an article are selected by the best subset selection method. This magnifies the importance of article freshness in predicting the virality of an article.

Table 2.4: The $AdjR^2$ value and list of the important features extracted from each feature set using the best subset selection method (all articles).

	$AdjR^2$	**Important Features**
Temporal	70.3	Page View Count and Normalized Page View Count time-series
Content	70.9	Top ten stories page view, Top ten stories content similarity, Title length, Probability of content neutrality and compoundness, and SMOGIndex
Social	72.5	Tweet time difference, #Followers in the first 30 minutes, Retweet and Normalized retweet time-series

Table 2.5: The R^2 value and list of the important features extracted from each feature set using the best subset selection method (viral articles).

	R^2	**Important Features**
Temporal	72.6	Page View Count and Normalized Page View Count time-series
Social	77.3	Tweet time difference, #Followers in the first 30 minutes, Retweet and Normalized retweet time-series
Content	79.4	Topic intersection, Top ten stories page view, Top ten stories content similarity, Title length, and Content neutrality

2.2.4 Deployment at The Washington Post

Throughout this section, we explored various features and trained a regression model for the popularity prediction task. In order to help journalists and editors at The Washington Post, we deployed this model and built a real time forecasting system for each article. Once a news article is published, the forecasting system begins to track it and extract features described in Table 2.1. With the help of Splunk [5], we built a dashboard to order articles based on their forecasted popularity. As we make predictions for news articles, we also track the actual page views of articles for evaluation purposes. In addition to providing popularity prediction to editors and journalists, we are also able to ascertain how well our proposed regression model performs over the latest news articles. We use the same setup to evaluate the performance of the forecasting model. We collect all articles that have been published in August 2015 (after the forecasting system is deployed). Table 2.6 summarizes the performance of this model. Although the deployed model is not using all the proposed features, it has a comparable performance to the results shown in Table 2.3.

[5] http://www.splunk.com/

Table 2.6: Evaluation of the deployed model for articles published in August 2015

Data	$AdjR^2$	Top 1% (R^2)
August 2015	0.829	0.620

2.3 Predicting the Shape and Peak Time of News Article Views

Most everyday content that we experience in our online lives exhibit a bursty life-cycle, e.g., stock quotes [140], posts on social media such as Twitter and Facebook [78], and viral videos on YouTube [1]. For our purposes here, a burst can be characterized as a brief period of intensive activity followed by a long period of nothingness [119]. Explanations of bursty behavior revolve around a threshold behavior [55] wherein, for instance, people share or read about an article once a handful of his or her friends share or read that article. This reinforcing behavior around an object leads to a burst of activity. In modern news agencies, peak prediction could be used to decide whether to move a specific article reaching to its maximum number of viewers, to the top of the main page or similarly move it to lower parts of the page once it loses traction.

Our focus here is to determine when an object of interest will likely reach its maximum views (or shares or likes). This is different from prior studies which focused primarily on either aggregate prediction or burst detection. Aggregate prediction is the task of estimating the total (but not peak) number of views or the total size of a cascade [6, 23, 46, 69, 110, 133]. Burst detection aims to detect (but not predict) the burst time as it happens [49, 88].

Shape and peak time prediction in the context of news articles is very challenging for at least two reasons. First, traditional regression models cannot be directly applied due to the quick rise-and-fall pattern of bursts [70]. Secondly, different articles exhibit significantly distinct lifespans. For instance, as we will observe later in Section 2.3.2, sport articles have much shorter lifespans than political articles in The Washington Post dataset.

In this section, we propose two approaches to predict peak time for news articles. First, we adopted a classification method similar to [124]. We use the same set of features listed in Table 2.1 and train several classifiers to do the prediction with them. However, results of this approach are not good enough. We then present a better alternative online approach. The online method that we propose combines a time-series clustering method called KSC [129] and a time-series detection method called $SpikeM$ [70] for the purpose of peak and shape prediction for news articles. Our evaluation shows that this online approach significantly outperforms classification approach in all settings.

2.3.1 Related Work

Related work can be grouped into two categories: burst time detection and burst time prediction.

In burst time detection, we are given the entire time series and our goal is to spot bursts in it. This area of research is very prominent in the literature as there are significant applications, e.g., in telecommunications, wherein we aim to detect network anomalies using bursts in the number of packages sent/received, in financial time series analysis, wherein we aim to spot sharp price fluctuations in the stocks [140], or in detecting web trends [125]. Bursts in such time series are typically detected by dividing the time series into equal or variable windows and detecting the one that exhibits an anomaly [140].

In burst time prediction, the problem is, given the current time series, to predict whether we will witness a burst in the near future or not. Traditionally, classical linear methods such as auto regression (AR) and moving average (MA) [73] have been utilized for this purpose. However, as stated earlier, the quick rise-and-fall property of practical time series makes such approaches inadequate and necessitates the development of better models. One of the well-known approaches to modeling bursts is the work of Kleinberg [49], wherein he suggested to model the stream using an infinite state automaton so that bursts correspond to state transitions. Bursts associated with state transitions constitute a nested structure where a long burst of low intensity potentially contains several bursts of higher intensity inside it, in a recursive way. Using this method, Parikh *et al.* [88] proposed a method to rank the bursts in an E-commerce query setting and classify them based on their shape. In a similar work by Wang *et al.* [124], the burst detection problem is modeled as a binary classification problem wherein if a time series has a burst after a specific time μ, its respective class would be positive and negative, otherwise. To predict the exact burst time of the time series, the author suggests an iterative solution and classification for every desired point in time. This is a serious drawback since every potential burst point in a time series requires a new classifier to be trained. We show that the traditional way of predicting burst by classifying to binary classes does not necessarily provide the best result and we demonstrate how our proposed method overcomes these limitations.

2.3.2 Time Series Shape Identification

In this section, we describe two approaches to model the shape and detecting the peak time of a news article. Our first model is a classification approach that is inspired by a previous work [124]. Then, we will present a better online alternative for this task.

It is easier to predict the burst time of the article once we have an initial understanding of the pattern of views for a news article. In order to predict the pattern of click counts for a news article, we use a combination of clustering and classification models. We first cluster all the time series in our training dataset. Then, we consider these clusters as class labels

and use metadata and contextual features that are extracted from a news article before its publication to classify them. We adopted the clustering method KSC [129] to cluster time-series of training news articles into four clusters. Note that other clustering methods such as K-Shape [86] could be used for this purpose, as well but KSC offered a better clustering according to our experiments. The choice of number of clusters is reached using Silhouette testing [100], in which a clustering with four clusters reached the highest score.

Fig 2.4 represents the cluster center for these four clusters along with a pie chart representing the distribution of burst time for articles in each cluster. The time-series represented in Fig 2.4 shows an exponential rise and power low fall which is inline with the result of previous works [70, 109, 129]. The exponential rise in the time-series is the reason why the burst time is leaned toward the left side of the plot.

The clusters are ordered so that C1 contains the most number of articles and C4 contains the least number of articles. Each cluster has its own characteristics and represents a set of specific articles:

- **Cluster 1**: C1 represents articles that have an extreme bursty behavior. More than 42% of articles in our dataset fall into this cluster, which shows that most of the news article have a very short lifespan and are only popular in a very short period of time. These news article are mostly Sports and National related topics. It is surprising that 70% of these articles have their peak four hours after their publication and only 17% have their peak before 30 minutes. This means that although there is a bursty behavior in the viewing pattern of these articles, despite the expected behavior, this bursty behavior will not happen right after the publication of news.

- **Cluster 2**: C2 represents articles that have a sudden burst and follow-up views by people. More than 74% of the articles in this cluster have their peak within 2 hours after publication. These are stories that are likely to go viral very rapidly and people talk about it even days after their burst time. These articles are mostly blog articles and usually talk about Politics and Opinions.

- **Cluster 3**: C3 represents articles that become popular, gradually. No article in this cluster has its peak within one hour after its publication. Almost 78% of these articles have their peak after four hours of their publication. This behavior is completely aligned with the shape of the cluster. These are the stories that go popular slowly and lose their popularity at a similar pace. These news content are almost equally divided to blogs and articles on almost all topics. People will gain interest in blog articles after a good amount of people talk about it and that is the reason for the slow increase of number of views in the beginning.

- **Cluster 4**: C4 has a similar behavior to C1, with a small difference that there is a small follow-up after the peak. These articles covers Sports, National, and Foreign topics and usually are more trending than the news article in C1. More percentage of

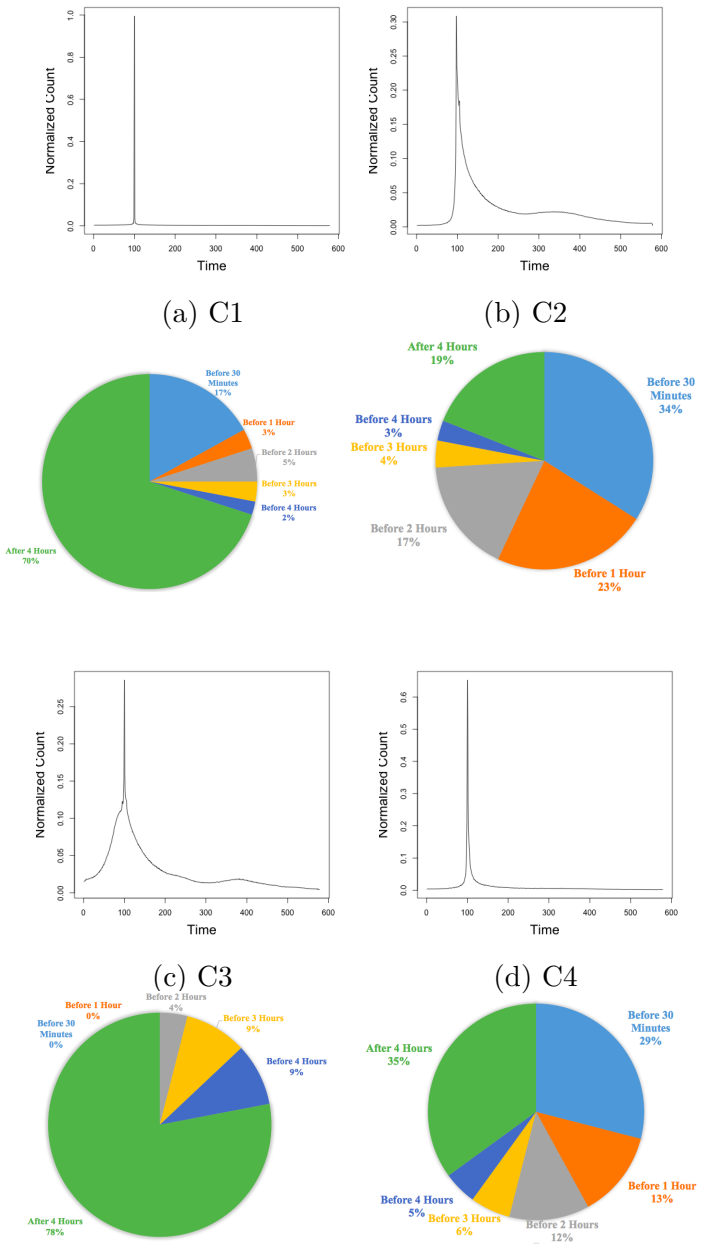


Figure 2.4: Different cascading pattern of news articles along with the pie chart representing the portion of news articles that have their peak before a specific time for each cluster.

articles have their peaks in the first 30 minutes than those in C1 cluster. At the same time, there are fewer articles to peak 4 hours after their publication.

From Fig 2.4, the life-span of a news article could be captured by these four clusters. We can predict the shape of viewing pattern of a news article by classifying it into one of these four

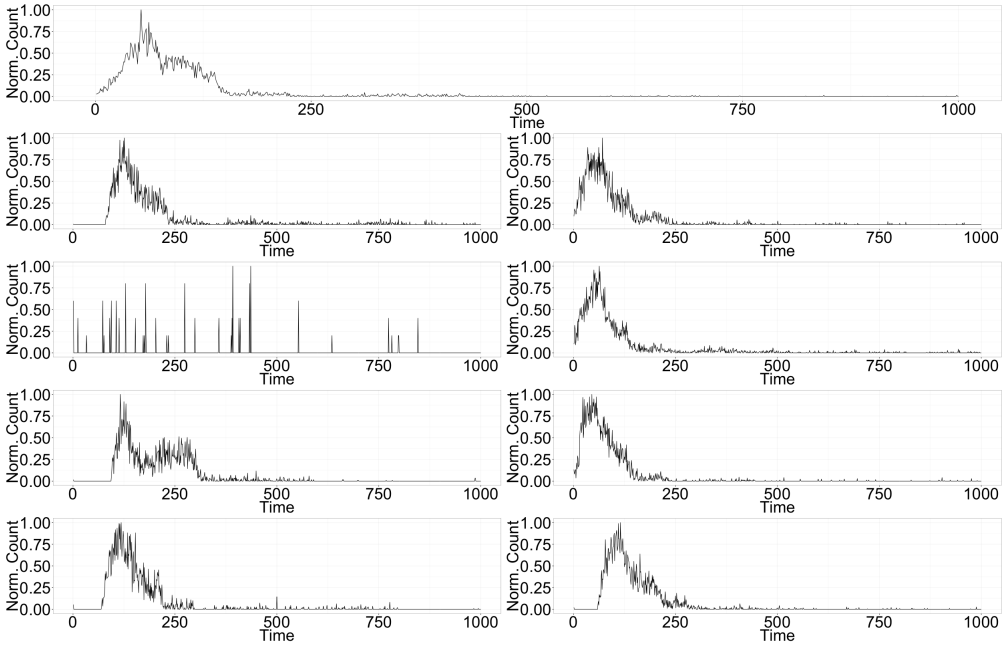


Figure 2.5: An example of the queried article (the most top plot) along with the plot of time series for its top eight most similar articles.

Table 2.7: Result of shape prediction by classifying the news article using features that are extracted before its publication. For the classification, we only use metadata and contextual features since this classification is required to be done before publication of the article.

Precision	Recall	F-Score	Weighted Precision	Weighted Recall	Weighted F-Score
0.7	0.7	0.7	0.7	0.7	0.68

classes. Besides metadata and contextual features, we introduce some additional effective features based on an interesting observation.

Fig 2.5 enumerates 8 out of 10 most similar time-series to the given query news article. As you can see, the plot of most of these time series has a similar shape to the time series of the query article. We can use this resemblance between the time series of query article and its top ten similar articles to find the shape of the query article. The cluster number of these top ten most similar time-series are also used as additional features in the classification task.

Once we collect all features, we train a multi-class Random Forest classifier and use 10-fold cross validation to verify the performance on our data. The result is shown in Table 2.7. Table 2.8 reports the detailed Precision, Recall, and F-Score value for each cluster. As you can see in this table, we have a good classification result for the first three clusters, while a poor performance on cluster C4. As explained earlier, the reason for this poor performance is the abundance of all types of article in this category, which makes it hard for the classifier to distinguish the articles in this cluster from other clusters.

Table 2.8: The detailed performance of the shape prediction method representing how the model works for each cluster

Metric/Classes	1	2	3	4
Precision	0.67	0.67	0.73	0.66
Recall	0.73	0.56	0.94	0.28
F-Score	0.70	0.61	0.82	0.39

2.3.3 Offline Peak Time Prediction

In the previous section, we have identified four possible viewing patterns of a new article. Now, we move on to the second problem which estimates the approximate peak time of the news article. First, we treat this problem as a binary classification task. Wang *et al.* proposed CPB (Classifiers to Predict the Bursts) [124] which is a method that by using multiple classifiers in each time frame tries to predict whether we will have a peak in the μ future time windows. Therefore, the main prediction problem changes to a binary classification. The major problem with this method is that it generates a new classifier for each time frame and each value of μ. On the other hand, it collects temporal features and update them incrementally as they move along the time series of the news article. In our proposed method, we only use the features that are collected within the first 30 minutes after publication of an article. Although using data collected within the first 30 minutes after the publication of article might sound insufficient for the purpose of this problem, we show that our method can achieve good result even with this limitation. Another reason for having this limitation on our problem is the necessity to act as early as possible to possible changes that might be required for a news after receiving the result of peak prediction.

Our method could be considered as a simpler version of the CPB, however it is as effective as this method. We first explain CPB method and then explain our model in terms of this model. Each time-series in CPB is divided into K time windows and it extracts various temporal features using these K time windows. A spike is the $f(k_{max})$ that satisfies $\forall\ 1 \leq k \leq K,\ f(k_{max}) \geq f(k)$, where k_{max} is the time window of the spike and $f(.)$ is the number of page views at any given time interval. Given two constants K and μ and the observed cascade spreading in time interval $[t_0, t_{current}]$, where t_0 is the starting time of the cascade and $t_{current}$ is the current time, the μ_{th} future time interval is defined as $[t_{current} + \frac{\mu-1}{K} \times (t_{current} - t_0), \frac{\mu}{K} \times (t_{current} - t_0)]$. Given this definition, CPB tries to find whether we see a peak in the μ future time window. On the other hand, by solving the above problem incrementally, it is able to predict in which time window the burst will appear.

We create four different classifiers to predict whether we will have a burst within 30 minutes, within an hour, within two hours, and within three hours after publication of an article. In order to make the prediction for the first 30 minutes, we only use the contextual and metadata feature, while for the rest of the classifiers we use the whole feature set. We consider all the articles that have their peak before a specific time, as the positive class and all the other articles as negative class. Therefore, we are trying to have a high true positive while

Table 2.9: Result of predicting approximate time-frame where a peak would occur for an article using classification framework

	Precision	Recall	F-Score	#Pos
First 30 Minutes	0.19	0.99	0.31	3747
First One Hour	0.28	0.99	0.43	5512
First Two Hours	0.36	0.99	0.52	7201
First Three Hours	0.41	0.99	0.57	8151

reducing the false positive. Table 2.9 presents the result of our SVM classifiers for the peak prediction.

The recall value represents how well we can classify articles in the positive class. As you can see in this table, in all the experiment the recall value is close to maximum, meaning that if an article have a peak before that specified time, we will be able to report that. However, the low precision value shows that we also report lots of articles that have their peak after that specified time, as the ones that have their peak before it. Therefore, we have to try to reduce the amount of false positive rate in our classifiers. As you can see in Table 2.9, the precision value increases as we try to make prediction for a farther future. This is due to the fact that as we try to predict for farther future, each dataset turns to look like a balanced dataset and number of positive and negative samples tend to get closer. This means that rather dealing with a rare-class classification problem in the beginning, we are dealing with a balanced dataset in the future.

2.3.4 Online Peak Prediction

In Section 2.3.3, we described a classification method to predict the peak of news articles. This method is an offline method since the prediction is done once and model is not updated with changes of article behaviors. After exploring a rather comprehensive set of features with latest classification models, it is difficult to achieve a significantly better prediction results with this classification framework. Alternatively, we propose an online method. We track real time click time-series of each news articles after their publication and identify similar historical click time-series of previous articles and use them to predict shape and peak times for newly published news articles. In this method, we use the *SpikeM* algorithm [70] to predict whether we will have a peak in the future time intervals. Although *Spikem* is originally created for time-series detection, we are the first to use it for predicting peak time of news articles. In the following section, we shortly explain the *SpikeM* model and how we adapt this method to predict the shape and peak time of a news article.

SpikeM Algorithm

The *SpikeM* algorithm is designed to model the rise and fall in the time-series pattern of different events. This model can take into account time-series characteristics such as exponential rise and power-law fall pattern, periodicity, and avoiding divergence to infinity. *SpikeM* model uses a handful of parameters that each of them captures one of these characteristics. The main equation for detecting the pattern of the time-series is as follows:

$$\Delta B(n+1) = p(n+1).\Big(U(n).\sum_{t=n_b}^{n}(\Delta B(t) + S(t)).f(n+t-1) + \epsilon\Big) \tag{2.4}$$

Where $\Delta B(n)$ is the number of people who just clicked on the article, $U(n)$ is the number of uninformed people[6], $S(t)$ represents when the time-series peaks, ϵ is the noise parameter of the effect of external activities around the article which is usually $\epsilon \approx 0$, $f(t)$ is the amount of infectiveness of an article at time t which is defined as $f(t) = \beta \times t^{-1.5}$. As you can see $f(t)$ is a decaying function over time which means that as we move along time the infectiveness of the article also fades away. β represents the interestingness of the article and a low β value means that no one cares about this news. $p(n)$ captures the periodicity of the event and has the following equation:

$$p(n) = 1 - \frac{1}{2}P_a(\sin(\frac{2\pi}{P_p}(n + P_s) + 1) \tag{2.5}$$

The periodicity of the time-series is controlled using three parameters which are the strength of periodicity, P_a, the actual period, P_p, and phase shift of the periodicity P_s. The periodicity captures mainly the fact that readers tone down their activity during the night.

SpikeM uses the *Levenberg-Marquardt (LM)* to minimize the sum of the errors between the actual number of viewers at time n, $X(n)$ and the estimated $\Delta B(n)$:

$$D(X, \theta) = \sum_{n=1}^{n_d}(X(n) - \Delta B(n))^2 \tag{2.6}$$

where n_d is the duration of the time-series. We consider the *SpikeM* model as a smoothing algorithm that can be used to provide a better resolution of the time-series in our historical dataset. Through our experiments, we find out that this model provide a much better result than other smoothing algorithms such as LOESS smoother [25].

SpikeM for Shape Prediction

Matsubara *et al.* [70] explained how to use *SpikeM* algorithm in order to use it for forecasting the volume and pattern of the next time-intervals. According to this method, we have to wait until we see the first spike in a time-series to be able to predict the rest of the time-series. This comes from the fact that once an event like "Harry Potter" movies, shows a pattern in

[6] N represents total number of people who viewed the article, and $B(n) + U(n) = N$

the past, it pretty much shows the same pattern of user activities in the future. Therefore, using the spike patterns in the past, we would be able to infer *SpikeM* parameters and then use them on the new time-series to predict the future. Although, this hypothesis highly relies on the fact that similar events would entice the same behavior in user activities, it would not have any solution for events that does not have a history.

Therefore, in order to use the *SpikeM* model, we have to use the model in a way that it uses the pattern of similar articles in the past and use them for prediction. To solve this problem, we suggest two approaches to find the similar time-series to an article. As mentioned in Section 2.3.2, for each news article in our dataset, we extract top ten similar articles to it. As we argued in Section 2.3.2 and according to Fig 2.5, the viewing pattern of the queried time-series is quite similar to the viewing pattern of these top ten articles. Therefore, these are the strongest candidates for finding the similar pattern.

In the second approach, we find the similarity on all our historical dataset. This could be viewed as the method that is used by Google Correlate[7], which finds the most similar time-series to the input time-series in their dataset.

In both of these methods, we try to find the most similar article to the queried article. We later explain in this section different measures to identify the most similar time-series to the queried article. Having similar events with similar viewing pattern is all we need to use *SpikeM* algorithm to predict the future time-intervals of the queried article. Our solution on using the top ten articles and *SpikeM* consists of the following steps:

- For each queried article, we extract top ten similar articles to it and extract their time-series.
- For each of these ten time-series, we fit the *SpikeM* model and store their respective parameters.
- We use the parameters of these ten models to create ten time-series. As we move along the time, we compare the time-series of the current article with these ten time-series. We use four different measures to estimate the closeness of the each of these ten time-series to the queried article.
- Note that as we move along the time, for different time intervals, we find the most similar time-series. This is due to the fact that in first few hours after the publication of article, we see a lot of fluctuation in the article that is chosen as the most similar articles.

The above-mentioned approach could also be used for finding the most similar article in the whole historical dataset. In this method, rather finding the most similar article among only

[7] correlate.google.com

the top ten articles, we find the similarity of the queried article with the time-series in the whole dataset.

Using the above-mentioned procedure, we are able to have an online prediction for all articles that have at least one similar article in our historical dataset. Although it is possible that trend of the current article behave completely different from the historical ones, there is still a large number of articles that imitate the same behavior.

Similarity Measure

Time-series similarity metric is a very important component in our framework. Besides two methods that we used to measure time-series similarity in 2.3.4, we compare the performance of three different similarity measures in this paper. We use a normalized version of time-series in which we normalize the time-series by the maximum value that is been recorded up to the current time. We use Euclidean distance, the KSC similarity measure [129], and our proposed method which is a modified version of KSC method which we call Lagged KSC (LKSC) to compare time-series. These measures are briefly explained as follows:

1. ***Normalized Time-Series***
   Given the current time t and a time-series $\mathbf{x}_t$, the normalized time-series is calculated by $\hat{\mathbf{x}}_t = \frac{\mathbf{x}_t}{max(\mathbf{x}_t)}$. Therefore, given two normalized time-series $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{y}}_t$, the Euclidean distance between these time-series is calculated using $Euclid(\hat{\mathbf{x}}_t, \hat{\mathbf{y}}_t) = \sqrt{\sum_{i=1}^{t}(\hat{\mathbf{x}}_t^i - \hat{\mathbf{y}}_t^i)^2}$, where $\hat{\mathbf{x}}_t^i$ is the number of clicks at time point i.

2. ***KSC***
   As mentioned by Yang *et al.* [129], there are some problems in using the Euclidean distance to find the similarity between two time-series. We need a distance measure that is invariant to translation and scaling. Therefore, given two normalized time-series $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{y}}_t$, KSC defines the distance between them as:

$$d(\hat{\mathbf{x}}_t, \hat{\mathbf{y}}_t) = argmin \frac{||\hat{\mathbf{x}}_t - \alpha \hat{\mathbf{y}}_t^{(q)}||}{||\hat{\mathbf{x}}_t||} \tag{2.7}$$

   where $\hat{\mathbf{y}}_t^{(q)}$ is the result of shifting the $\hat{\mathbf{y}}_t$ time-series by q time units and $||.||$ is the l_2 norm. With a fixed value for q, α is considered as a convex function and we could find the optimal value of α by setting the gradient to zero which will gives us the following estimatino for α:

$$\alpha = \frac{\hat{\mathbf{x}}_t^T \hat{\mathbf{x}}_t^{(q)}}{||\hat{\mathbf{x}}_t^{(q)}||^2} \tag{2.8}$$

   Whereas given a fixed q, we can simply find the optimal value for α, there is no trivial solution to find the best value for q. Therefore, KSC shifts all the time-series to peak at a specific time unit and then use this distance measure to find the similarities

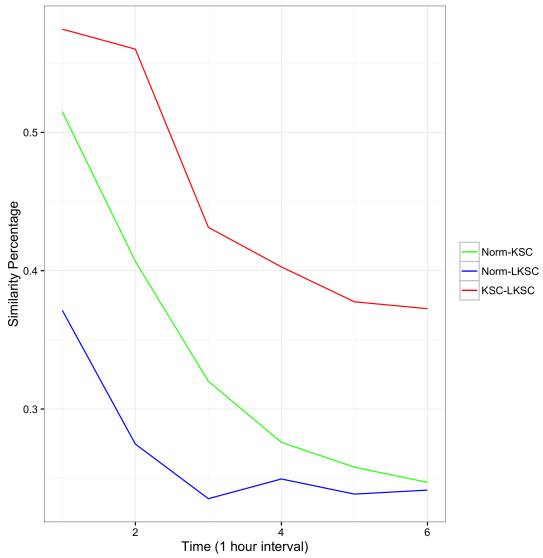


Figure 2.6: For each pair of similarity measures, this plot represents the percentage of article that have identical most similar article over each time interval

between time-series. However, in our problem, we are not aware of the peak time of the current article and therefore, we consider the maximum value that is seen so far as the peak value and use this value for normalization.

3. ***Lagged KSC***
KSC lacks in finding the best shift value q since we do not know the peak time of the current article. Therefore, we have to find a way to deal with finding the best value for q. As mentioned earlier, if we normalize the time-series of the current article according to the maximum value that is been recorded until now, we can use this method to shift the time-series and match peaks of the current article with its top ten time-series. In order to find the q value, we use the Cross Correlation between the normalized version of current time-series and its similar articles. The Cross Correlation method is a measure of similarity of two series as a function of lag of one relative to the others. The Cross Correlation between two series $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{y}}_t$ is measured as follows:

$$(\hat{\mathbf{x}}_t \star \hat{\mathbf{y}}_t)[n] = \sum_{m=-\infty}^{\infty} \hat{\mathbf{x}}_t^*[m]\hat{\mathbf{y}}_t[m+n]. \tag{2.9}$$

where $\hat{\mathbf{x}}_t^*$ denotes the complex conjugate of $\hat{\mathbf{x}}_t$ and n is the lag. In our application, $\hat{\mathbf{x}}_t$ is the current time-series while $\hat{\mathbf{y}}_t$ represents the time-series of similar articles. We find the best lag value that could maximize the Cross Correlation between two time-series and consider it as the value for q:

$$q = \arg\max_n(|(\hat{\mathbf{x}}_t \star \hat{\mathbf{y}}_t)[n]|) \tag{2.10}$$

Although both the KSC and LKSC will not provide a correct measure of the distance

between the two time-series (since we use a portion of time-series to compare them), it could be considered as an approximation for the distance between the two.

Since each of these measures capture a different aspects on similarity between two time-series, the result of the most similar article would also be different for these measures. To understand the amount of differences between these measures, for each pair of measures, we plot the percentage of articles that found an identical similar article. Fig 2.6 represents this percentage over each time interval[8]. As you can see in this figure, the similarity between the Normalized measure and LKSC is the least while KSC and LKSC has the largest similarity in the first 30 minutes. Moreover, once we go along the time, the similarity of all pairs decreases which magnifies on how different each of these measures behave when we compare enough number of data points from two time-series. On the other hand, Fig 2.7 compares the average distance of the queried article and its most similar article along the time. Fig 2.7 also plots the distance of the KSC and LKSC method when using the whole dataset to search for the most similar article. These plots are shown using KSC^* and $LKSC^*$, respectively. As you can see, the Euclidean distance, on the normalized time-series, gives the largest distance among other measures. As stated in [129], this confirms the inefficiency of the Euclidean distance for measuring the distance between two time-series. Moreover, our modification on the KSC method, Lagged KSC, gives a much smaller distance than KSC which shows how effective this approach is in finding the most similar article. Furthermore, LKSC provides a better distance than KSC^*. Please note that LKSC only searches among the top ten similar articles while KSC^* searches through the whole dataset to find the most similar article. Not surprisingly, $LKSC^*$ provides the least distance between the queried article and its most similar article which shows the effectiveness of this method among other measures.

To show the effectiveness of each of these measures in finding the most similar article, we plot the time-series of an article along with their most similar *SpikeM* generated time-series. Fig 2.8 represents the plots of time-series of an article over 24 hours. Each plot represents the time-series of the article along with the most similar *SpikeM* generated time-series to it according to a specific measure. We use the first 48 data points, representing the page views after four hours of publication of the article, to compare the two time-series. The left plots show the results using the Euclidean, KSC, and LKSC measures on the top ten similar articles, while the right plots shows the result of searching the whole dataset using KSC* and LKSC*. As you can see in this figure, the KSC method completely failed to find the similar article, while the Euclidean method finds a more realistic time-series to the trend of the article. Although Euclidean method found a similar time-series, it still fails to capture the invariant that exists between the two time-series. This problem is completely resolved by LKSC and LKSC* which capture both the trend of the article and the invariant that exists in the time-series.

Fig 2.8 represents the result of using different distance measures over 48 data points of the

[8]The first time interval represents the first 30 minutes, while all the other time intervals are separated with one hour. Therefore, time interval 6 represents the similarity for five hours after the publication

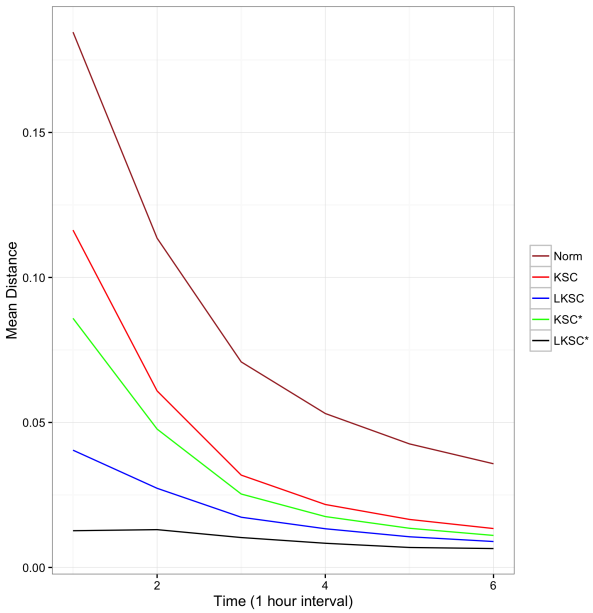


Figure 2.7: The average distance between the time-series of the queried article and its most similar article that is found by each measure over time

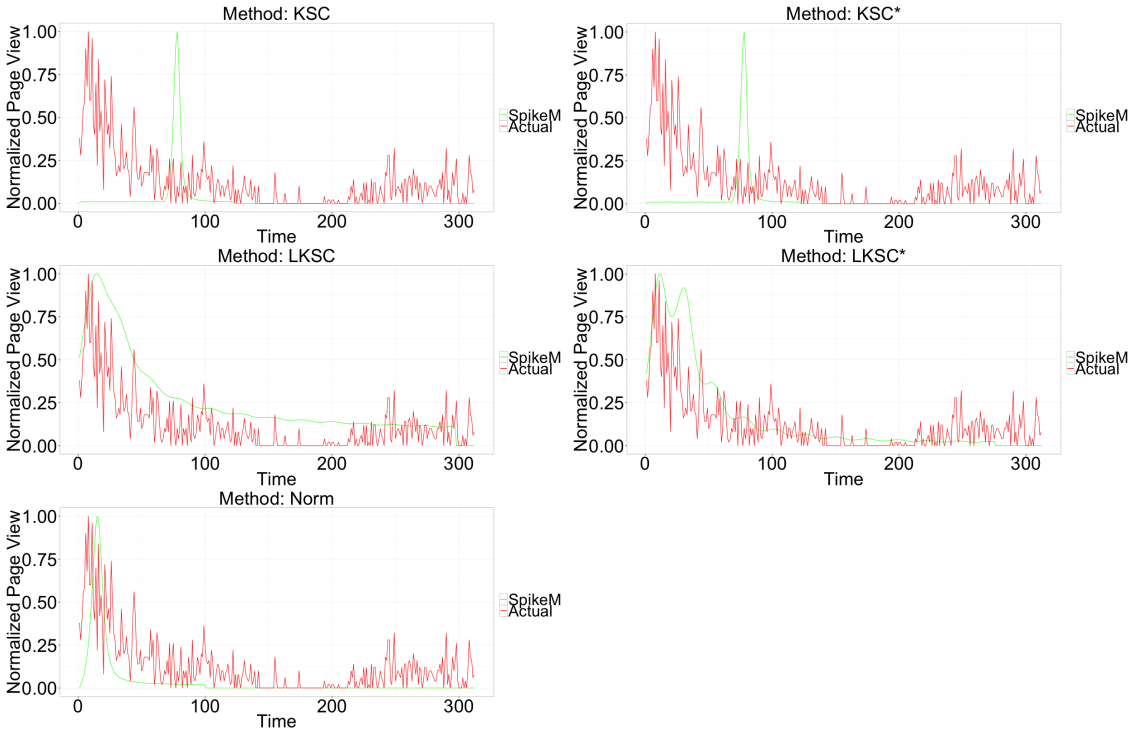


Figure 2.8: The result of finding the most similar article using KSC (top-left), LKSC (middle-left), Euclidean method (bottom-left), KSC* (top-right), and LKSC* (middle-right) by accommodating the first 48 data points

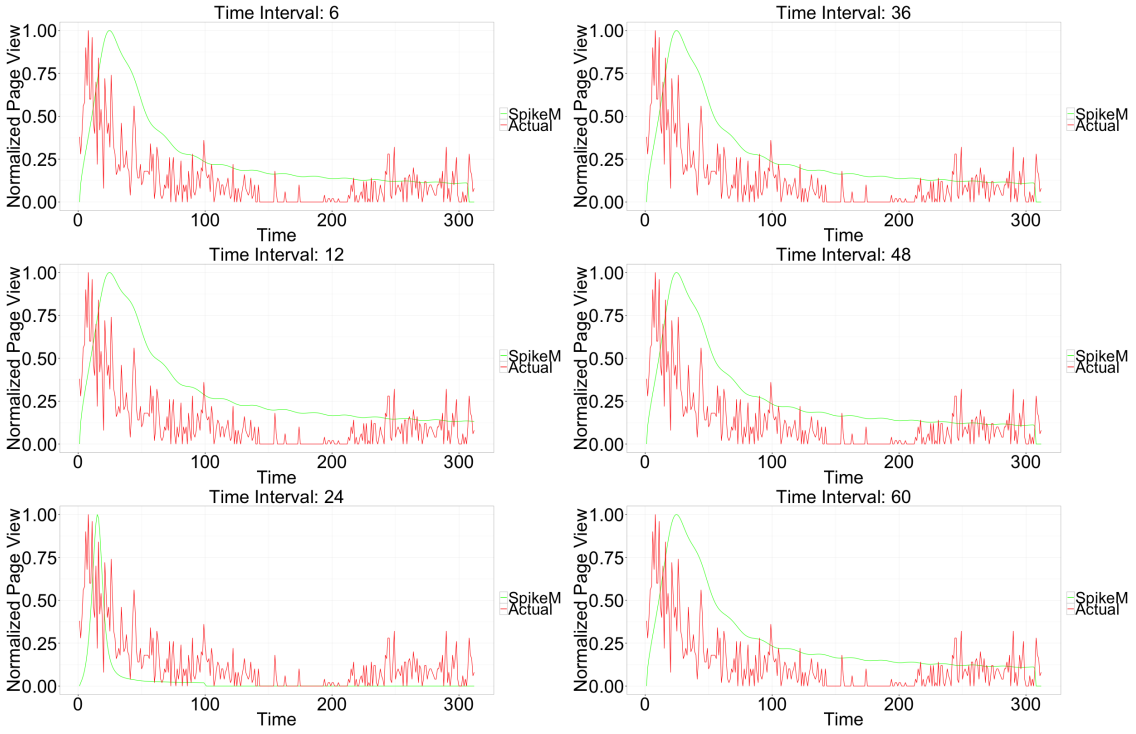


Figure 2.9: How most similar article will change over time, if we use LKSC measure to find it

Table 2.10: Result of using *SpikeM* to predict the peak

	Precision	Recall	Fscore	AUC
First 30 Minutes	0.48	0.99	0.65	0.76
First One Hour	0.47	0.99	0.64	0.77
First Two Hours	0.65	1	0.79	0.76
First Three Hours	0.68	1	0.81	0.75
First Four Hours	0.93	1	0.96	0.75

queried article and the *SpikeM* generated time-series. However, as mentioned earlier, we start finding the most similar article to a queried article after 30 minutes of its publication. Fig 2.9 shows how the most similar article will change as we move along the time and use more data points to find it. Fig 2.9 shows the result of using LKSC measure to find the most similar article.

As you can see, although just by using 6 data points (top-left plot) LKSC is able to find the right trend for the article, it lacks to shift the *SpikeM* generated time-series to match the data, correctly. As we move along the time, however we see that the *SpikeM* model that is similar to the time-series also changes to a narrower time-series that could better capture the trend and invariant in the time-series.

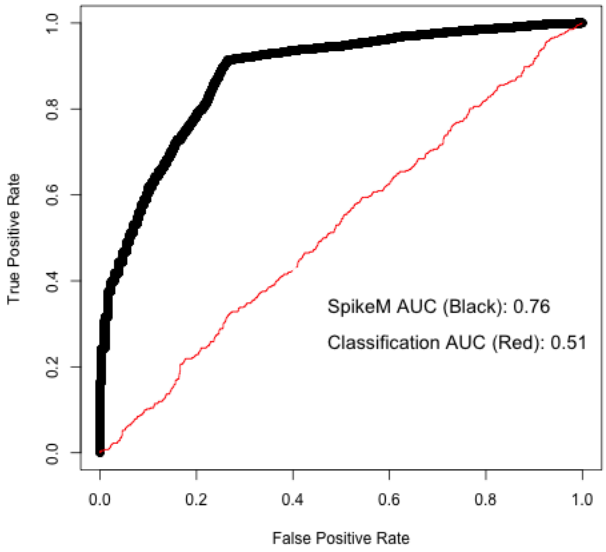


Figure 2.10: The comparison of the ROC curve for the classification and *SpikeM* method for the first 30 minutes.

Classification vs. Online

To compare the performance of the *SpikeM* method with the classification approach explained in Section 2.3.3, we use *SpikeM* with LKSC distance to predict the peak time of each time-series and compare the peak time with the actual peak time of the time-series. For a given time interval t (like 30 minutes), let us consider the prediction as p and the actual peak time as a. We create a binary label for prediction and actual peak as follows:

$$\begin{aligned} l_p &= 1 \text{ if } p <= t, \text{ otherwise } l_p = 0 \\ l_a &= 1 \text{ if } a <= t, \text{ otherwise } l_a = 0 \end{aligned} \tag{2.11}$$

Using these two binary vectors for prediction and actual peak time, we are able to compare our result to the classification result. Table 2.10 represents the results for the *SpikeM* method. As shown in this table, our proposed online approach significantly outperforms the classification framework (Table 2.9) in all settings. This shows nonlinear methods such as *SpikeM* is a much better alternative in predicting shape and peak times of news articles.

Fig 2.10 plots the ROC curve for the two approaches and compares the AUC value for these two methods. From the result, the classification method is no better than a random classifier, while *SpikeM* significantly performs better than this method.

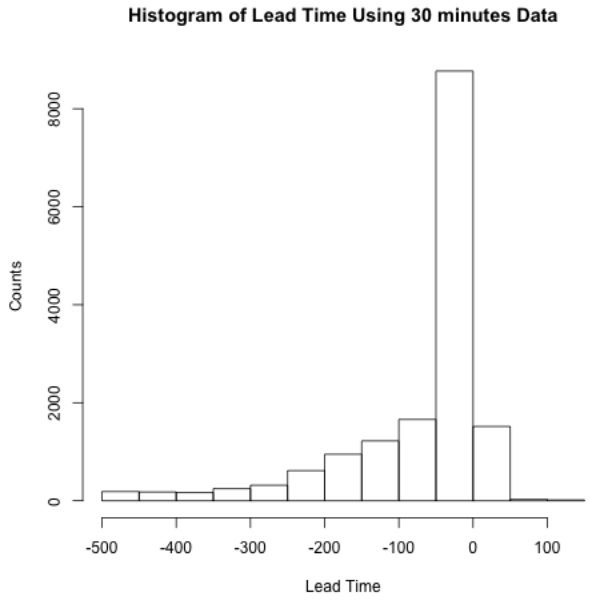


Figure 2.11: The lead time of online peak prediction for the first 30 minutes

Lead Time

As mentioned earlier, we use *SpikeM* not only to predict the shape of the time-series, but also to predict the peak time of the news article. Section 2.3.4 elaborates on how to predict the shape of the time-series using the *SpikeM* algorithm. One advantage of having a good method for shape prediction is that we can use the predicted pattern to find the peak time of the article. In this method, we consider the peak time of the most similar *SpikeM* generated time-series to a queried article, as our prediction for the peak time of the article. The *lead time* is defined as the time difference between the predicted peak time and the actual peak time. A positive lead time means that we predicted the peak time after the actual peak, while a negative lead time means that we predicted the peak before the actual peak. For all articles, we find the most similar *SpikeM* generated time-series to the pattern of the queried article according to the measures that are explained in Section 2.3.4.

Fig 2.11 represents the histogram of the lead time using this method for the first 30 minutes. The ideal shape for this histogram is to have as few articles as possible on the right side of zero point, while having as many articles as possible close to the zero point on the left side. As you can see in this figure, a large portion of news article has their lead time close to zero, which shows the effectiveness of *SpikeM* in predicting the peak time.

2.4 Conclusion

In this chapter, we explored different factors that play essential roles on the popularity of news articles, i.e., temporal, social, and contextual features. Our evaluation results show that among these three sets of features, the contextual features related to the freshness of an article are the most important factor in predicting the page views of viral articles, while metadata features are the strongest signals for predicting the performance of news articles in general. From these three sets of features, we also identified the most effective features that significantly contribute to popularity prediction of news articles. Motivated by the offline evaluation results, we deployed the model at The Washington Post.

Moreover, we introduced two problems on forecasting the pattern of click time-series and peak time of the news articles. We proposed offline and online methods to predict the shape of the click time-series before and after publication of an article. While, we use a combination of clustering and classification method to provide a solution for peak time prediction problem. Through our analysis on a real news dataset collected from The Washington Post, we provided a framework that could be used to both forecast the pattern of click time-series and peak time of the news article.

Chapter 3

Actor-Critic Model for Text Summarization

Text summarization is the process of summarizing a long document into few sentences that capture the content of the entire document. These summaries are usually comprised of a few sentences that briefly describe the abstract information in a text. For instance, in most of the current news agencies' web portals, after the news headline, these summaries play an important role in capturing the attention of readers.

Traditional text summarization models usually relied heavily on language-specific models and compression based techniques [27] to select the most salient sentence in the document as a representative for the summarization of an article. However, in recent years and with the advent of deep learning, neural network models are widely used for this task. Most of the current work in this area could be divided into two different categories: (1). *Extractive summarization* where, similar to compression-based models, the goal is to find sentences from the original document that are best representative of the entire article [80]. (2). *Abstractive summarization* where, the goal is to generate short and grammatically correct sentences that accurately captures the content of an article [105]. The underlying framework for most of the recent summarization techniques is a sequence-to-sequence (seq2seq) framework where the input article is processed by a sequence of encoders and the output is passed to a series of decoder units which generate the final summary [47]. There are various ways to design the overall summarization framework using these encoder/decoder units. Most of the studies in text summarization work with a word-level encoder/decoder units where each unit in encoder/decoder will take/generate a single word [24, 101]. However, recent models that work with sentence-level representation have shown more promising results [21, 82]. In these models, each unit in encoder/decoder will process/generate a full sentence from the original article. Therefore, most of these models solely focus on extractive summarization and offer no solution for abstractive summarization. On the other hand, with recent progress in using Reinforcement Learning (RL) for seq2seq problems, it is known that using RL-based techniques during training of a text summarization model will improve the result of these models [90]. Recent attempts in proposing a consistent extractor-abstractor framework fail in terms of generating better abstractive result for this task. Recently, Fast-RL model [21] proposed the first joint extractor-abstractor framework for the abstractive summarization that used Q-Learning technique [126] for training the joint model. However, according to their results, the extractor module trained with the RL technique achieves far better results

than the combination of extractive and abstractive model. Strictly speaking, the abstractor proposed by this study which is a simple pointer-generator model, will worsen the results generated by the extractor.

In this section, we propose a new text summarization model that will not only work at the sentence-level to select the most salient sentences in the document (extractive summarization) but also paraphrase these sentences using a RL-based language model to achieve abstractive summarization. Our model comprises of two different modules that are pre-trained first and used by a third module to provide the final framework: (i) **Extractor**, which selects the most salient sentences from the document using a classification and ranking model and (ii) **Abstractor**, which changes these sentences to provide the abstractive summarization and finally (iii) the co-training mechanism based on Generalized Advantage Estimation (GAE) technique [104] in RL that connects these two different modules. Using this technique will not only allow us to achieve a more robust and coherent training but also it reaches the state-of-the-art (SOTA) result on CNN/Daily Mail [40] and DUC 2004 [1] text summarization datasets. Our contributions for this work are three-fold:

- We propose a new extractor-abstractor framework model for the task of abstractive text summarization which utilizes a hierarchical representation for sentence and document representation and uses this representation to extract the most salient sentence from the original article. Moreover, the abstractor module is trained using a self-critic policy gradient technique.

- We propose an RL-based co-training algorithm for the extractor-abstractor module based on Generalized Advantage Estimation (GAE) technique which provides robust convergence w.r.t. simple Q-learning. To the best of our knowledge, this is the first work that applies GAE algorithm for training the critic network in text summarization problems.

- Through our extensive experiments, we show that our proposed model achieves SOTA result on abstractive text summarization for CNN/Daily Mail and DUC 2004 datasets.

3.1 Related Work

There is a vast amount of research work on the topic of text summarization using deep neural networks. These works range from fully extractive methods [17, 21, 22, 82, 83, 97, 134, 139], compression-based methods [57, 123] to completely abstractive ones [36, 52, 105].

[1] https://duc.nist.gov/data.html

3.1.1 Extractive Summarization

As one of the earliest works on using neural networks for extractive summarization, Nallapati *et al.* [80] proposed a framework that used a ranking technique to extract the most salient sentence in the input. In the past years, similar frameworks were used for extractive summarization that try to classify or score sentences in the document based on their saliency [22, 81, 130]. Moreover, some of these works utilize the classifier score for each sentence to rank sentences or use other techniques to re-rank the result of a classifier. Recently, Narayan *et al.* [82] used this idea to first rank sentences based on their classification score and then used a RL-based algorithm to re-rank these sentences based on the reward that the generated summary receives w.r.t. the ground-truth data. Zhou *et al.* [139] tackled this problem by learning a scoring model that, after reading the input, extracts sentences one after another during training. In this work, they proposed using the intermediate reward achieved after extracting each sentence and unlike prior works, which used cross-entropy loss, they used KL-divergence as the objective function for training their model.

3.1.2 Abstractive Summarization

For abstractive summarization, Rush *et al.* [101], for the first time, used attention for a seq2seq model for the problem of headline generation. To further improve the performance of these models, pointer-generator model [79, 121] was proposed for successfully handling the Out-of-Vocabulary (OOV) words. This model was later improved by using the coverage mechanism that avoids generating repetitive words in the summary [105]. Although pointer-generator model could solve the OOV problem in abstractive text summarization and is able to generate semantically correct sentences, it still tends to copy large portions of the original article as the summary. To fix this issue, models such as bottom-up abstractive summarization [36] that utilizes a content selector to select only relevant parts of the source document are recently proposed. Currently, this model holds the SOTA result for abstractive text summarization on CNN/DM dataset. However, all these models, suffer from a common problem known as "*exposure bias*" which refers to the fact that during training, the model is trained by feeding ground-truth input at each decoder step, while during the test, it should rely on its own output to generate the next token. Also, training in these models is typically done using cross-entropy loss, while during testing, metrics such as ROUGE [66] or BLEU [87] are used to evaluate the model. To tackle this problem, researchers suggested various models based on scheduled sampling [10] and Reinforcement Learning [19, 21, 52, 61, 90]. Paulus *et al.* [90] improved upon the pointer-generator model [105] and proposed using a temporal and intra-attention model during decoding. They also proposed the first self-critic policy gradient approach for training a text summarization model. However, RL-based models, as suggested by these works, usually take a lot of time to converge and are very slow during training.

3.1.3 Extractive-Abstractive Summarization

Recently, several authors have investigated methods which try to first perform extractive summarization by selecting the most salient sentences within a document using a classifier and then apply a language model or a paraphrasing model [63] on these selected sentences to get the final abstractive summarization [21]. Our work also falls into this category. The best performing model in this category is the Fast-RL model [21] which uses an attention-based model for the extractor to select the most salient sentences in the article while they use a simple language model to paraphrase the sentences selected by the extractor. Finally, they use a critic network for co-training the extractor-abstractor and use Q-Learning for training this critic network. However, as mentioned before, according to the results obtained by this method, adding the abstractor module to the co-training model will greatly affect the performance of their model.

3.1.4 RL-based model for Text Summarization

Recently, there is a surge in using RL-based techniques in boosting the performance of deep neural networks [47, 62]. RL-based frameworks are widely used in computer vision applications [4, 75], however, there are few works that utilizes these models for NLP tasks [94]. Although there have been great success stories in using various RL-based models for robotic applications [62] and computer games [76], most of the research works in NLP applications only use policy gradient-methods and Actor-Critic frameworks. Among the policy-gradient based models, REINFORCE [127] algorithm and Self-Critic Policy-Gradient method [98] have been widely used in various NLP applications [42, 51, 64, 84, 132] including text summarization [82, 90, 122, 128]. On the other hand, in Actor-Critic based models [19, 39, 59], Q-Learning [126] and Double Q-Learning [117] are the algorithms that are primarily used for training the critic network in text summarization [21, 61]. Our method uses an Actor-Critic framework for text summarization, however for training our critic network, we use a more robust method known as Generalized Advantage Estimation (GAE) method [104].

3.2 Proposed Method

Our proposed model comprises of three different modules that are separately trained. The extractor unit is trained such that it will select sentences which contain more salient information w.r.t. to the article. The abstractor unit is trained to paraphrase a sentence and provide a better abstractive result for each sentence. Finally, our extractor-abstractor co-training model will use Reinforcement Learning (RL) to connect these two different modules and provide the final model.

Let us assume that our dataset is comprised of document and ground-truth summary pairs

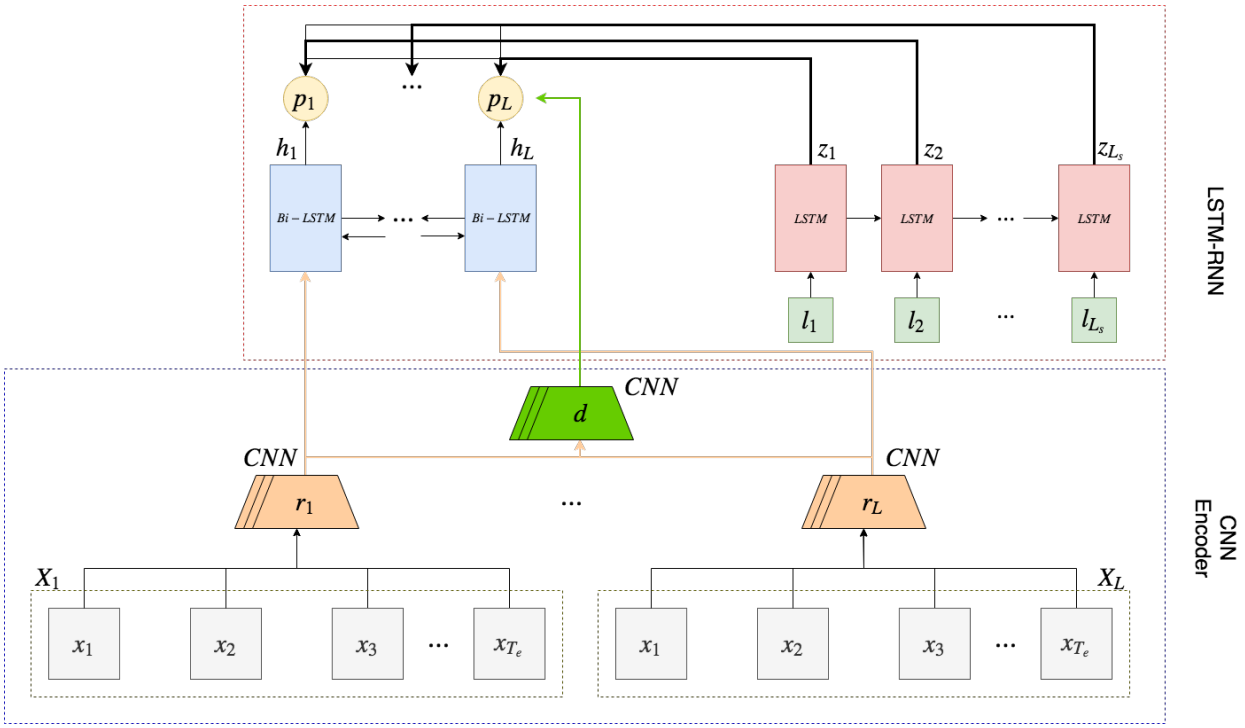


Figure 3.1: Our proposed extractor network is comprised of a two-layer CNN encoder (bottom blue block) that provides the high-level sentence and document representation for the LSTM-RNN encoder-decoder framework (top red block) that captures the long-term dependency of the sentences and finally selects sentences.

given by $\{D_i, G_i\}_{i=1}^{N}$. For the rest of the paper, we will discard the subscript i whenever it is evident that we are considering a single article. Each document is represented by a series of sentences, $D = \{X_1, X_2, \cdots, X_L\}$, where L is the number of sentences in the article. Similarly, the ground-truth summaries are represented by a series of sentences, $G = \{Y_1, Y_2, \cdots, Y_{L_s}\}$, where L_s is the number of sentences in the summary. Each sentence in the article and summary are represented by a series of words, $X_i = \{x_1, \cdots, x_{T_e}\}$ and $Y_j = \{y_1, \cdots, y_T\}$, where T_e is the length of article sentences while T is the length of a summary sentence and x, y are words within a sentence and words that are generated by the abstractor, respectively. The goal of extractor is to train a function to find a mapping $f : D \rightarrow \hat{D}$ such that $\hat{D} \subseteq D$. On the other hand, the abstractor will try to learn a mapping $\hat{f} : \hat{D} \rightarrow G$ such that $\hat{f}(f(D)) = G$. Given this formulation, the extractor will find a subset of sentences from the original document that are relevant for summary, while the abstractor will paraphrase these sentences to provide the final abstractive summary for the entire document. Table 3.1 summarizes all the notation used in this paper.

Table 3.1: Notations used in this section.

	Notation	Description
General Parameters		**Document Representation**
	D	A document containing a list of sentences ($D = \{X_1, \cdots, X_L\}$)
	$\hat{D}$	A subset of sentences from the original document, $\hat{D} \subseteq D$
	L	Number of sentences in the input document
	G	Ground-truth summary sentences ($G = \{Y_1, \cdots, Y_{L_s}\}$)
	$\hat{G}$	Generated summary from our model ($\hat{G} = \{\hat{Y}_1, \cdots, \hat{Y}_{L_s}\}$)
	L_s	Number of sentences in the ground-truth summary
		Sentence Represetation
	X_i	A sentence containing a list of words ($X_i = \{x_1, \cdots, x_{T_e}\}$)
	T_e	Length of a sentence in the article
	Y_j	A ground-truth summary sentence containing a list of words ($Y_j = \{y_1, \cdots, y_T\}$)
	$\hat{Y}_j$	A summary sentence generated by abstractor containing a list of words ($\hat{Y}_j = \{\hat{y}_1, \cdots, \hat{y}_T\}$)
	T	Length of the summary sentences
	V	A vocabulary of words
	x	A word in the original document, $x \in V \cup \{OOV, EoS, SoS\}$
	y	A word in the summary sentence
	$\hat{y}$	A word generated by the model
		CNN Encoder
	k	Number of filters used by the CNN
	r_i	CNN sentence representation for $r_i = CNN(X_i, k)$
	d	Document representation for the whole document $d = CNN(\{r_i, \cdots, r_L\}, k)$
Extractor		**LSTM Encoder**
	$\overrightarrow{h_i^{ext}}$ $\overleftarrow{h_i^{ext}}$	Forward and backward hidden state of the encoder: $h_i^{ext} = LSTM(r_i, h_{i-1}^{ext})$
		RNN Decoder
	z_j^{ext}	Hidden state of the decoder, $z_j^{ext} = LSTM(l_j, z_{j-1}^{ext})$
	a_{ij}^{ext}	Unnormalized attention score for calculating the context vector
	α_j^{ext}	Normalized attention score for calculating the context vector
	c_j^{ext}	Context vector
	b_{ij}^{ext}	Unnormalized probability distribution over sentences
	p_j^{ext}	Probability distribution over sentences
Abstractor		**LSTM Encoder**
	$\overrightarrow{h_i^{abs}}$ $\overleftarrow{h_i^{abs}}$	Forward and backward hidden state of the encoder: $h_i^{abs} = LSTM(x_i, h_{i-1}^{abs})$
		LSTM Decoder
	$\hat{Y}$	A sentence generated by the abstractor using greedy selection ($\hat{Y} = \{\hat{y}_1, \cdots, \hat{y}_T\}$)
	Y'	A sentence generated by the abstractor using sampling selection ($Y' = \{y'_1, \cdots, y'_T\}$)
	z_j^{abs}	Hidden state of the decoder, $z_j^{abs} = LSTM(y_j, z_{j-1}^{abs})$
	a_{ij}^{abs}	Unnormalized attention score for calculating the context vector
	α_j^{abs}	Normalized attention score for calculating the context vector
	c_j^{abs}	The context vector
	b_{ij}^{abs}	Unnormalized probability scores over words in the dictionary
	p_j^{abs}	Probability distribution over words in the dictionary
Actor-Critic		**Critic Network**
	R	Reward function used by critic network
	$\hat{R}$	Discounted reward
	γ	Discount parameter
	V_ψ	Critic estimation of the reward-to-go for a sentence
	A_ψ	Advantage estimation of the reward for a sentence
	s_t	Current state of the actor
	λ	Trade-off parameter for GAE that handles variance and bias
		Training Parameters
	θ	Extractor training parameters
	ϕ	Abstractor training parameters
	φ	Actor training parameters
	ψ	Critic training parameters

3.2.1 Extractor

The extractor is trained such that it will find the most salient sentences in the original document. The extractor has an encoder which reads the full document and a decoder which will select important sentences from the article.

Extractor Encoder

Due to the recent successes on using hierarchical models in capturing high-level information when reading a long document [35], we also design the encoder with a temporal Convolutional Neural Network (CNN) [48]. Through using various-length kernels, CNN models are known to be great at capturing these high-level contextual data for a long document. Specifically, our extractor is comprised of two CNN layers (lower blue box in Fig 3.1), where the first CNN layer reads words' representation in a sentence to provide a sentence representation and the upper layer uses these sentence representations to provide the document representation. The basis of our hierarchical encoder is similar to most of the CNN-based encoder models designed for various seq2seq problems [33] and it is mostly similar to framework used by Chen *et al.* [21], with the primary difference being that their extractor only relies on sentence representation while we also utilize the document representation as an important signal for the extractor to select salient sentences. Fig 3.1 shows our hierarchical encoder-decoder model for the extractor. The bottom part shows our two-layer CNN encoder which provides the sentence and document representation for the upper layers. The temporal CNN will read the document word-by-word and create a sentence representation $r_i = CNN(X_i = \{x_1, \cdots, x_{T_e}\}, k)$, while the second CNN layer reads these sentence representations and generate a single vector $d = CNN(\{r_1, \cdots, r_L\}, k)$ representing the full document representation, where k is the number of filters used by the CNN. Moreover, since CNN models lose temporal information of a document, we use another LSTM-based encoder which tries to capture the long-term sequential dependency of sentences in the document. Specifically, this series of Bi-Directional LSTM encoders will read the sentence representation r_i generated by the CNN model and generate a forward and backward hidden state $\overrightarrow{h_i^{ext}}$ and $\overleftarrow{h_i^{ext}}$ for sentence i which is later on used by the decoder during sentence selection ($\overrightarrow{h_0^{ext}} = \mathbf{0}$ and $\overleftarrow{h_L^{ext}} = \mathbf{0}$). For simplicity, we use h_i^{ext} to represent the concatenation of $\overleftarrow{h_i^{ext}}$ and $\overrightarrow{h_i^{ext}}$, i.e., ($h_i^{ext} = \overrightarrow{h_i^{ext}} \oplus \overleftarrow{h_i^{ext}}$). Therefore, through using this CNN-LSTM encoder, we not only keep the high-level context of the document, but also add the ability to capture long sequence dependencies between the sentences for the model.

Extractor Decoder

We use a series of RNN units to represent our decoder. During decoding, each decoder will receive the index of the ground-truth target sentence l_j along with the previous decoder state

Table 3.2: The comparison of the oracle extractive summarization score on CNN/DM test dataset.

Oracle Extractor	**R-1**	**R-2**	**R-L**
ROUGE-L F-score	51.92	29.33	49
ROUGE-L Recall	48.4	27.72	45.28

z^{ext}_{j-1} ($z^{ext}_0 = \mathbf{0}$) and generate a hidden state z^{ext}_j and attend twice to the original sentences in the document to find the final scoring over each sentence.

$$z^{ext}_j = RNN(l_j, z^{ext}_{j-1}) \tag{3.1}$$

Given this, to find the sentence that is selected at time j, we use a pointer-generator model [121] to attend over the sentences from the original document and use z_j and document representation d to select a sentence. Adding the document representation d will help the model to always focus on selecting sentences that are closer to the context of the entire document.

$$\begin{array}{lcl} a^{ext}_{ij} & = & v_1 tanh(W_1 h^{ext}_i + W_2 z^{ext}_j + W_3 d) \\ \alpha^{ext}_j & = & softmax(a^{ext}_j) \\ c^{ext}_j & = & \sum_i \alpha^{ext}_{ij} W_1 h^{ext}_i \end{array} \tag{3.2}$$

where a^{ext}_{ij} will be the unnormalized attention scores for each sentence in the document based on the output state of the decoder z^{ext}_j and the document representation d. We will then use a softmax operation to turn these scores to attention scores and use the hidden state of the LSTM encoder h^{ext}_i to create a context vector c^{ext}_j over these sentences. Finally, we will use this context vector to create the probability distribution for each sentence in the document as follows:

$$\begin{array}{lcl} b^{ext}_{ij} & = & v_2 tanh(W_4 h^{ext}_i + W_5 c^{ext}_j) \\ p^{ext}_j & = & softmax(b^{ext}_j) \end{array} \tag{3.3}$$

where p^{ext}_j is the probability distribution over all sentences in the document at step j of sentence selection and all W's and v's are model parameters.

Training Extractor

Each article in our training dataset is accompanied with summary sentences. Therefore, in order to provide the extractor with right target labels during training, we follow the same strategy used by Chen *et al.* [21]. To find ground-truth target labels for our extractor, we find the closest sentence in the original article for each summary sentence. However, unlike Chen *et al.* [21] which used ROUGE-L Recall score between the ground-truth summary sentence and sentences in the original document, we use ROUGE-L F-score to select these labels. The motivation behind choosing ROUGE-L F-score rather than Recall is shown in Table 3.2. As shown in this table, choosing ground-truth label according to the F-score will

result in a much stronger oracle summary than the one generated with Recall. Therefore, each ground-truth label l_j for extractor is generated as follows:

$$l_j = \arg\max_i ROUGE - L_{F-score}(X_i, Y_j) \tag{3.4}$$

Given these labels, the extractor is finally trained using cross-entropy loss as follows:

$$\mathcal{L}_{extractor} = \sum_{j=1}^{L_s} -\log p_\theta(l_j | z_j^{ext}, c_j^{ext}) \tag{3.5}$$

where p is the output distribution from Eq. (3.3) and θ represents all the training parameters for extractor. During training of the extractor, we manually zero out the score of a previously selected sentence by setting its unnormalized score b_{ij} to $-\infty$. This will avoid generating the same sentence during the extraction. This is a non-differentiable operation and will be avoided later during our extractor-abstractor co-training stage using reinforcement learning. Moreover, similar to any other seq2seq models, we will use a specific token to signal the decoder regarding Start-of-Selection (SoS) and End-of-Selection (EoS) and therefore the decoder learns to generate EoS token whenever it is done with sentence selection.

3.2.2 Abstractor

The abstractor module will generate abstractive summarization for each sentence that is selected by the extractor. The abstractor model could be built by either a simple language model [21] or a complex paraphrasing model [63]. Though, a simple abstractor model could generate paraphrases of a sentence fast, it has a tendency to generate poor paraphrases [21]. On the other hand, a complex paraphrasing model (like [63]) will require a lot of time for training and would be harder to merge with the final co-training algorithm. Therefore, in our proposed model, we use a simple yet effective paraphrasing model which uses the same pointer-generator model proposed by Paulus *et al.* [90]. This model was proposed for text summarization task, however, we develop a simpler version of it without any intra-attention and temporal attention during decoding since we are only using it for paraphrase generation.

Abstractor Framework

Our abstractor is a pointer-generator model that is trained using Self-Critic Policy-Gradient to create paraphrases for each sentence. Similar to the extractor, which uses attention over sentences, our abstractor attends over words in a sentence to generate the final output. This is an LSTM-LSTM framework in which the encoder reads a single sentence and the decoder will generate the paraphrase of that sentence. We use a bidirectional encoder and each encoder reads the word representation of x_i and generate a forward and backward hidden state $\overrightarrow{h_i^{abs}}$ and $\overleftarrow{h_i^{abs}}$ ($\overrightarrow{h_0^{abs}} = \mathbf{0}$ and $\overleftarrow{h_{T_e}^{abs}} = \mathbf{0}$) which we concatenate them and represent it with

$h_i^{abs} = (\overrightarrow{h_i^{abs}} \oplus \overleftarrow{h_i^{abs}})$. During decoding, the decoder receives the last state of the backward and forward encoder states ($z_0^{abs} = (\overleftarrow{h_0^{abs}} \oplus \overrightarrow{h_{T_e}^{abs}})$) along with the ground-truth word y_j and generate the output as follows:

$$\begin{aligned} z_j^{abs} &= LSTM(y_j, z_{j-1}^{abs}) \\ a_{ij}^{abs} &= v_1 tanh(W_1 h_i^{abs} + W_2 z_j^{abs}) \\ \alpha_j^{abs} &= softmax(a_j^{abs}) \\ c_j^{abs} &= \sum_i \alpha_{ij}^{abs} W_1 h_i^{abs} \\ b_{ij}^{abs} &= v_2 tanh(W_4 h_i^{abs} + W_5 c_j^{abs}) \\ p_j^{abs} &= softmax(b_j^{abs}) \end{aligned} \tag{3.6}$$

where p_j^{abs} is the probability distribution over all words in our dictionary and all W's and v's are model parameters.

Once we reach either the End-of-Sentence token or maximum decoding length T, we will calculate the loss using cross-entropy (CE) as follows:

$$\mathcal{L}_{abstractor}^{CE} = \sum_{j=1}^{T} -\log p_\phi(y_j | z_j^{abs}, c_j^{abs}) \tag{3.7}$$

Abstractor Training

We use the labels generated for our extractor training in Section 3.2.1, to create paraphrasing pairs for our abstractor. Therefore, $X = \{x_1, \cdots, x_{T_e}\}$ represents a sentence in the original document, while $Y = \{y_1, \cdots, y_T\}$ represents its respective ground-truth summary extracted using $ROUGE - L$ F-score in Eq. (3.4). Similar to other works which use Policy-Gradient for improving the performance of a seq2seq model, we first pre-train the abstractor using Eq. (3.7) and then activate our Self-Critic Policy-Gradient training. During Policy-Gradient training, for each decoder unit, we select two different sequences, one through sampling the output p_j^{abs} which is denoted by y'_j and another through greedy selection over p_j^{abs}, $\hat{y}_j = \arg\max p_j^{abs}$. Let us assume $Y' = \{y'_1, \cdots, y'_T\}$ as the sentence generated from sampling the output distribution and $\hat{Y} = \{\hat{y}_1, \cdots, \hat{y}_T\}$ as the sentence generated by greedy selection and R is our reward function. Therefore, the Self-Critic Policy-Gradient (SCPG) objective function could be defined as follows:

$$\mathcal{L}_{abstractor}^{SCPG} = \sum_{j=1}^{T} -\log p_\phi(y'_j | z_j^{abs}, c_j^{abs})(R(\hat{Y}, Y) - R(Y', Y)) \tag{3.8}$$

The reason this model is called a Self-Critic Policy-Gradient is due to the use of greedy selection as the baseline for lowering the variance of the sample selection reward. Although we could use a critic network to estimate the baseline reward, we choose to use the greedy sentence as the critic due its stability during training. Note that during Policy-Gradient

training, we completely remove the dependency on the ground-truth tokens and use the sampled sentence to calculate the loss. Moreover, we multiply this loss with the reward that we receive as the difference of the greedy and the sampled sentence. This will encourage the model to create sentences that are closer to the greedy selection and penalizes those that are worse than the greedy selection. We choose $ROUGE-L$ F-score as our reward function following the same intuition used for extractor. Finally, during Policy-Gradient training, we use a combined loss between the SCPG objective and CE objective to train the abstractor as follows:

$$\mathcal{L}^{Combined}_{abstractor} = (1-\eta)\mathcal{L}^{CE}_{abstractor} + \eta\mathcal{L}^{SCPG}_{abstractor} \tag{3.9}$$

where $\eta \in (0,1)$ provides the trade-off between the cross-entropy loss and SCPG loss. During training of the abstractor, we first pre-train the model with CE loss and then activate the combined loss with $\eta = 0.98$.

3.2.3 Extractor-Abstractor Co-Training

Once the extractor and abstractor are pre-trained, we need to connect these two modules to generate the final summary. For doing this, we create a copy of all the pre-trained trainable variables in both modules and train a joint model using Reinforcement Learning (RL). The extractor will select sentences from the original document while the abstractor paraphrases the sentences selected by the extractor to generate the final summary. To train our extractor-abstractor module, we use a technique in RL called Actor-Critic learning. Generally, in RL, the goal of an agent (the extractor-abstractor module in our case) interacting with an environment is to maximize the expectation of the reward that it receives by taking a particular action. In our problem, we want to maximize the expectation of the reward for generating a summary using our extractor-abstractor module, i.e.,

$$Maximize\ \ \mathbb{E}_{\hat{Y}\sim p_\varphi(\hat{Y})}[R(\hat{Y}, Y)] \tag{3.10}$$

where p is our extractor-abstractor agent and φ represents all the training parameters of this joint model. Fig 3.2 illustrates this framework. In our Actor-Critic co-training, the actor (the red block) receives a document and passes it to the extractor-abstractor agent and generates the output for the critic network. The critic network (the blue block), on the other hand, will provide feedback for the actor of how well the actor summarized the original document and the actor will improve itself based on the feedback it received from the critic.

Actor Model

The actor receives document D and generates the output summary $\hat{G} = \{\hat{Y}_1, \cdots, \hat{Y}_{L_s}\}$. We then assign a reward for the generated summary sentence at time t as follows:

$$R(t) = ROUGE - n_{Recall}(\{\hat{G}\}_0^t, \{G\}_0^{L_s}) \tag{3.11}$$

where $ROUGE-n$ Recall score is the proportion of number of overlapping n-grams in the generated output up to time t and the number of n-grams in the entire ground-truth summary. Once the actor generates the complete summary for an article and collects the rewards according to Eq. (3.11), we need to train the critic network based on these rewards.

Critic Network

In Actor-Critic learning, the critic network is a regression model that estimates the reward-to-go of the summaries generated by the actor. Although we can use the rewards as shown in Eq. (3.11) to train the critic network, in practice, we usually use either intermediate rewards such as $\hat{R}(t) = R(t) - R(t-1)$ [139] or discounted reward $\hat{R}(t) = R(t) + \gamma R(t+1)$ [47], where γ is the discounting parameter. In our proposed method, rather than raw rewards in Eq. (3.11), we use the normalized discounted reward, due to its stability during the training of the critic network. Let us assume that the current state of the actor is denoted by s_t. We define a function called advantage function that captures the difference between the normalized discounted reward (Q) and the reward estimated from the critic network (V_ψ) as $A_\psi(\hat{Y}_t, s_t) = Q(\hat{Y}_t, s_t) - V_\psi(s_t)$, where ψ represents the training parameters of the critic network. The critic network will receive the same sentence representation that the extractor uses to select sentences, i.e., $s_t = z_t^{ext}$ in Eq. (3.1) and estimate a reward for using this sentence as part of the output summary. This way of calculating the advantage function is called Q-Learning which suffers from having large variance and over-estimation of the critic network [117]. To avoid this problem, we propose using a method called Generalized Advantage Estimation (GAE) which provides a better trade-off between the bias and variance of the critic network during training. In GAE, we define the advantage function by expanding the Q-function as follows:

$$A_\psi^{GAE}(\hat{Y}_t, s_t) = \sum_{i=t}^{T} (\gamma\lambda)^{i-t} \Big(\underbrace{R(t) + \gamma V_\psi(s_{t+1})}_{Q(\hat{Y}_t, s_t)} - V_\psi(s_t) \Big) \tag{3.12}$$

where R is the mean-normalized reward, and $V_\psi(s_{t+1})$ and $V_\psi(s_t)$ show the estimates of the critic network for the next and current sentence, respectively. λ controls the trade-off between the bias and variance such that large values of λ yield to larger variance and lower bias, while small values of λ do the opposite. Once we estimate the advantage function using Eq. (3.12), we will multiply this estimate to the cross-entropy loss acquired by the actor to obtain the complete objective function that needs to be minimized:

$$\nabla_\varphi \mathcal{L}_\varphi = \mathop{\mathbb{E}}_{\hat{Y} \sim p_\varphi} [\nabla_\varphi - \log p_\varphi(\hat{Y}_t | s_t) A_\psi^{GAE}(\hat{Y}_t, s_t)] \tag{3.13}$$

Once the actor network is trained using this objective, the critic network V_ψ is trained using Minimum Squared Error (MSE) (just like any other regression model) as follows:

$$\mathcal{L}_\psi = (V_\psi(s_t) - \hat{R}(t))^2 \tag{3.14}$$

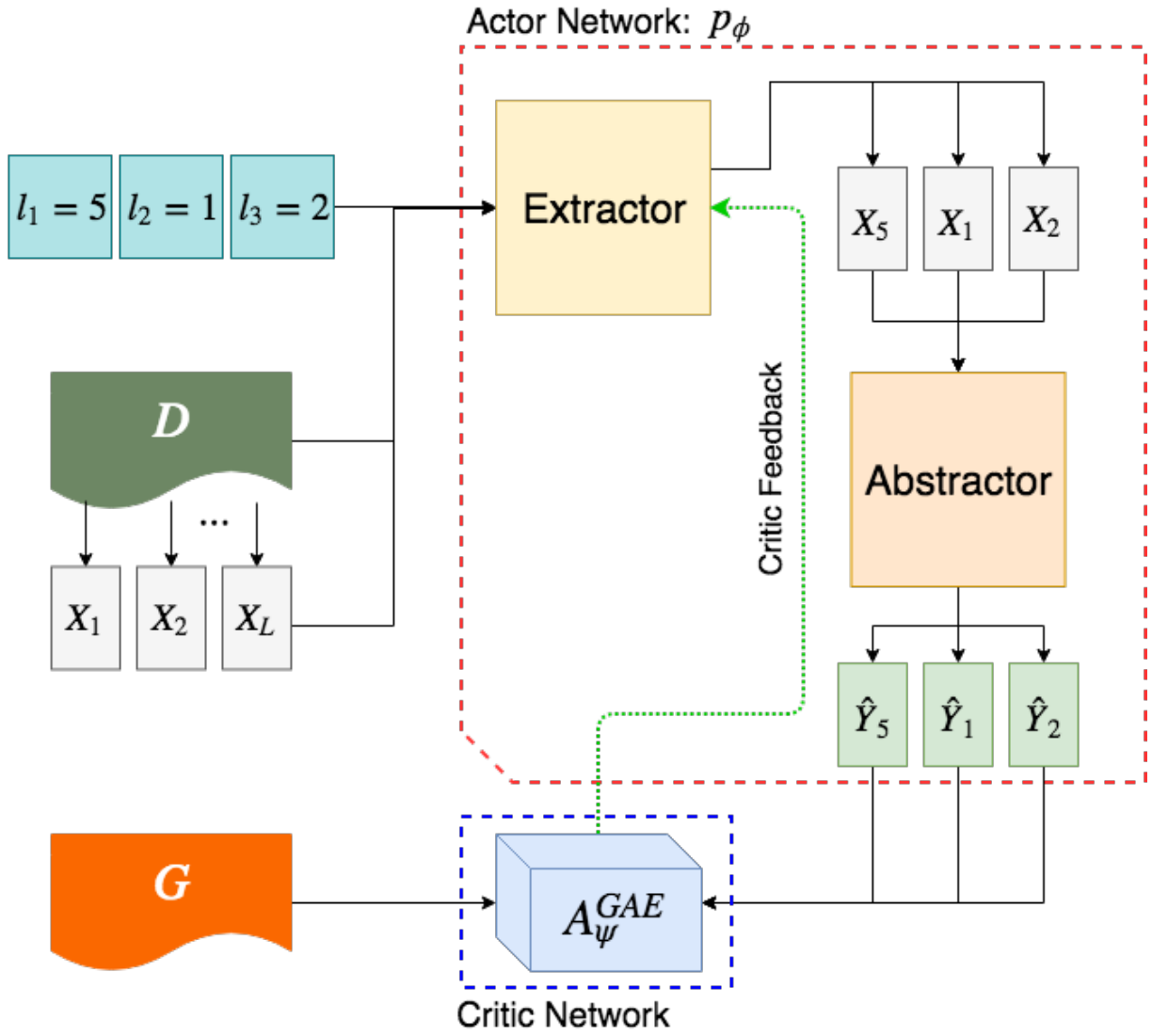


Figure 3.2: The overall framework of our proposed Actor-Critic model. The actor (the dashed red block) is comprised of our extractor-abstractor module which receives a document and generates a summary while the critic (the blue block) receives the ground-truth summary and the output generated by the actor and provides feedback as of how well the actor summarized the document.

where $\hat{R}$ is the normalized discounted reward and training continues until the actor model converges.

3.3 Experimental Results

We have run various experiments to evaluate the performance of our proposed model and compared it to the most recent extractive and abstractive test summarization models. As

mentioned in Section 3.2.3, we use ROUGE-n Recall score to calculate the true reward gained from the model. All experiments are performed using a P100 GPU and training of the extractor, abstractor, and the joint extractor-abstractor model takes only a few hours [2]. For evaluation, we use the ROUGE [66] and METEOR [8] metrics and compare our results in terms of ROUGE-1 (R-1), ROUGE-2 (R-2), and ROUGE-L (R-L) F-score. ROUGE score captures the number of shared n-grams between the generated summary and the ground-truth summary and provides Precision, Recall, and F-score values. While, METEOR uses the harmonic mean of the unigram Precision and Recall, weighted such that the Recall score carries a higher weight in the final score and is known to provide a better evaluation w.r.t. human-written summaries. For all models, we run the training as long as there is an improvement in terms of ROUGE-1 F-score on validation data and after five attempts of no improvement in the result on validation data, the training procedure is terminated. We will pre-train the extractor and abstractor, separately and then use the pre-trained model to run our co-training reinforcement learning model.

3.3.1 Datasets

Although the WP dataset that we used during the popularity prediction problem contained news article content, however this dataset did not have any ground-truth summary and collecting these summaries is very time-consuming and expensive. Therefore, to test our proposed model for this task, we used two of the well-established and well-studied open-source datasets for text summarization.

For all our experiments, we use the non-anonymized version of CNN/Daily Mail (CNN/DM) since it is widely used by the researchers for evaluating the text summarization models [40][3]. The CNN/DM dataset contains more than 300K news article which is accompanied by 3.76 human-written summary sentences, on average. We use the same train/validation/test split as used by See *et al.* [105] to train our model which leaves 287K, 13K, and 11K training, validation, and test articles, respectively. As shown in Table 3.2, the oracle extractive summarization on this dataset could only achieve 51% in terms of ROUGE-1 F-score which shows that most of the summaries in this dataset have an abstractive nature. Fig 3.3 shows the distribution of the extracted sentences using our best performing model compared to the ground-truth sentences extracted using Eq. (3.4). As shown in this figure, most of the sentences extracted by our model are from the first few sentences in the document while the ground-truth labels have a more diverse distribution than the proposed model.

Moreover, to test the generalization of our proposed framework, we evaluate our model using DUC 2004 [4] dataset. This dataset, which was part of the DUC summarization challenge,

[2]We used the same hyperparameters used by Chen *et al.* [21] for fair comparison.

[3]Following the setup of the pointer generator model, we used the following code to prepare the data: https://github.com/abisee/cnn-dailymail

[4]https://duc.nist.gov/data.html

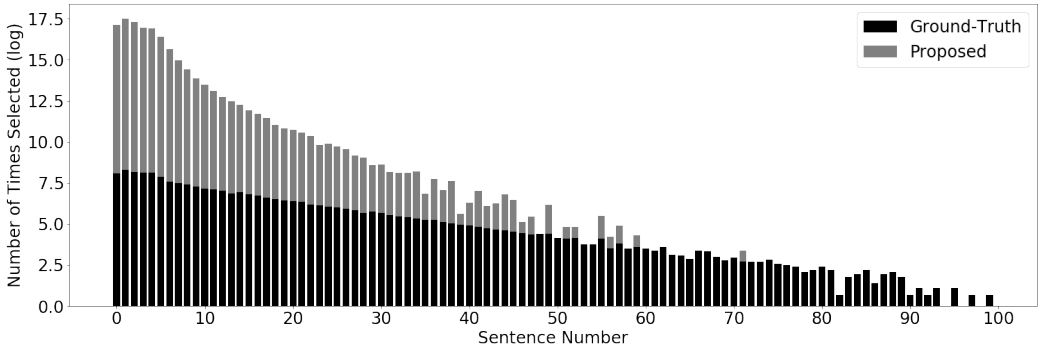


Figure 3.3: The distribution of the extracted sentences from our proposed model and the ground-truth on CNN/DM dataset. The number of sentences extracted by our model and ground-truth on this stacked bar-plot is based on the log-scale for better illustration.

contains only 500 articles and is only used for evaluation purposes. Each article in this dataset is accompanied by four human-written summaries and the goal is to have a model that can generate very short summaries, i.e., 10 words or 75 bytes. This challenge, however, evaluates the performance of each model based on the ROUGE Recall score gained at 75 bytes. Therefore, for this dataset, we report the ROUGE Recall score of our proposed method at the cut-off point of 75 bytes.

3.3.2 Extractive-Abstractive Summarization

To fully understand the performance of our Reinforcement Learning framework and our extractive, abstractive, and our joint model, we run separate experiments for each module. We first run various experiments to understand how our extractive summarization coupled with the GAE training will work on this dataset. Therefore, for this experiment, we are not using the abstraction module. We then add the abstractor module and run our GAE co-training algorithm on the extractor-abstractor module. This way, we will be able to comprehend how our abstractive model can enhance the performance of the extractor.

Extractive Summarization

Table 3.3 compares the performance of our RL-based extractive summarization with the most recent state-of-the-art (SOTA) models in text summarization. The SOTA result for extractive summarization belongs to the Fast-RL model [21] which is shown in the table using an underline. As shown in this table, our best performing model (the last row in this table) has a very close performance to Fast-RL model. According to the p-value analysis with $p < 0.01$, each experiment must have at least +0.25 improvement over the baseline to be considered as statistically significant. Thus, according to this analysis the improvement that

Table 3.3: Results of our RL-based extractive summarization on CNN/DM dataset.

	R-1	**R-2**	**R-L**	**METEOR**
Lead-3	40.29	17.59	36.53	-
Cheng & Lapata [22]	35.5	14.7	32.2	-
SummaRuNNer [80]	39.6	16.2	35.3	-
Refresh [82]	40.0	18.2	36.6	-
Fast-RL (ext+RL) [21]	41.47	18.72	37.76	22.35
Ext	35.32	15.17	31.79	22.54
Ext + GAE (n=1)	37.58	17.84	34.82	**25.31**
Ext + GAE (n=2)	38.98	18.30	36.00	24.69
Ext + GAE (n=3)	40.08	18.67	36.92	24.15
Ext + GAE (n=4)	40.12	18.68	36.96	24.01
Ext + GAE (n=5)	41.30	18.86	37.67	22.14

we see in Fast-RL is not significant compared to our proposed model. However, similar to this method, we are beating all the other extractive models in terms of ROUGE and METEOR metrics. As shown in this table, our best performing model for extractive summarization uses ROUGE-5 Recall score to calculate the reward in Eq. (3.11) and smaller n-grams will worsen the overall performance [5]. However, when we use METEOR as the evaluation metric, our best performing method is the method which uses ROUGE-1 Recall which beats the best existing model by almost 3%.

Abstractive Summarization

For this experiment, we use our entire co-training model. Therefore, given a document, the extractor selects the most salient sentences and abstractor will paraphrase them. Table 3.4 shows the result of our proposed model and compares them to the most recent models that hold SOTA in abstractive summarization. As shown in this table, our model beats the SOTA result in all metrics and have the runner-up result on ROUGE-2 metric. One of the most important aspect of our result in Table 3.4 is that unlike Fast-RL, which performs worse after applying abstraction to the extractor, we see consistent improvement over the corresponding extractive summarization in Table 3.3 using our model. This can be seen in this table by comparing the result of Fast-RL (ext+abs+RL) vs the Fast-RL (ext+RL). To better compare the behaviour of our proposed abstractor with Fast-RL model, table 3.5 shows the improvement gain that we receive in terms of R-1, R-2, R-L scores after adding the abstractor module to the extractor in Fast-RL and our proposed model. As shown in this table, the Fast-RL model has a negative improvement after adding the abstractor module, while our best performing model achieves more than 2.5% and 3% improvement in terms of R-1 and R-L scores, respectively.

[5] Our experiments with larger values of n showed no improvement.

Table 3.4: Results of our RL-based extractive-abstractive summarization on CNN/DM dataset with various state-of-the-art methods for abstractive text summarization.

	R-1	**R-2**	**R-L**	**METEOR**
Nallapati *et al.* [79]	35.46	13.3	32.65	-
Tan and Wan [111]	38.1	13.9	34	-
See *et al.* [105]	39.5	17.3	36.4	18.72
Fan *et al.* [32]	39.75	17.29	36.54	-
Paulus *et al.* [90]	39.87	15.82	36.9	-
Kryściński *et al.* [52]	40.19	17.38	37.52	-
ROUGESal + Ent + RL [89]	40.43	18.00	37.10	20.02
Hsu *et al.* [41]	40.68	17.97	37.13	-
Fast-RL (Ext + Abs + RL) [21]	40.88	17.8	38.54	20.38
Bottom-up [36]	41.22	**18.68**	38.34	-
Ext + Abs	37.45	14.3	35.78	19.28
Ext + Abs + GAE (n=1)	41.14	17.91	38.64	21.28
Ext + Abs + GAE (n=2)	**41.51**	18.32	**39.01**	**21.46**
Ext + Abs + GAE (n=3)	41.49	18.32	38.95	20.42
Ext + Abs + GAE (n=4)	41.37	18.23	38.82	20.24
Ext + Abs + GAE (n=5)	41.15	17.96	38.51	19.75

Table 3.5: Comparison of the improvement gain achieved after adding abstractive module to the extractor.

	Improvement Gain		
	R-1	**R-2**	**R-L**
Fast-RL (Ext + Abs + RL) [21]	-0.59	-0.92	0.78
Ext + Abs + GAE (n=2)	**2.53**	**0.02**	**3.01**

According to Table 3.5, in all experiments, our abstraction consistently improves the result of the extractor. Note that, during extractive summarization, we see continuous improvement on the result by increasing the n-grams that are used for the reward function. However, this behavior is not observed during the abstraction and our best performing model uses ROUGE-2 Recall for the reward in Eq. (3.11).

On the other hand, according to the METEOR score in this table, our best performing model is better than the SOTA result by more than 1%. Also, note that, in general, the METEOR scores received after abstractive summaries are lower than the extractive ones. This is not surprising since METEOR will judge the quality of a summary by how closely it is written to a human summary. Therefore, since during extractive summarization we only select sentences from the original document, the result will be much closer to human-written summaries.

Table 3.6: Comparison of the output of our proposed model with Fast-RL.

ARTICLE: **-lrb- cnn -rrb- thousands of palestinians are trapped in the devastated yarmouk refugee camp in syria, which has mostly been seized by groups including isis, activists report.** the london-based syrian observatory for human rights says isis and the al qaeda-affiliated al-nusra front took control of 90 % of the camp in southern damascus. <removed 1 sentence> **"never has the hour been more desperate in the palestine refugee camp of yarmouk", the statement said.** the unrwa estimates 18,000 civilians remain trapped in the camp that has been engulfed in fighting between the government and rebel forces since december 2012. syria's state-run sana news agency reports up to 2,000 people have fled in the past two days as food, water and medical supplies remain scarce. <removed 3 sentences>. the syrian observatory for human rights reports barrel bombs were dropped on the camp sunday as clashes continued. <removed 1 sentence> **"reports of kidnappings, beheadings and mass killings are coming out from al-yarmouk, which is under a brutal campaign of murder and occupation", palestine liberation organization executive committee member dr. saeb erekat said saturday.** <removed 5 sentences>
REFERENCE: isis and other rebel groups control most of the refugee camp, activists say. "never has the hour been more desperate in the camp", u.n. says. "reports of kidnappings, beheadings and mass killings," plo official says.
Fast-RL: **thousands of palestinians are trapped in yarmouk refugee camp in syria.** the london-based syrian observatory says isis and the al-qaeda-affiliated al-nusra front took **control of 90 %**. the unrwa estimates 18,000 civilians remain trapped in the camp. syrian observatory for human rights reports barrel bombs dropped on camp sunday.
Our Model: **thousands of palestinians are trapped in syria's yarmouk refugee camp.** isis and the al qaeda-affiliated al-nusra front took control of 90 % of the camp in damascus. **"never has the hour been more desperate in the camp", the statement said.** 2,000 people have fled in the past two days as food, water and medical supplies remain scarce.

A Sample Example

To compare the power of our proposed model with Fast-RL in terms of selecting the best sentences and generating abstractive summaries from those sentences, we selected one of our test samples [6] and compared the summary generated this method and our method. Table 3.6 illustrates this result. This article is comprised of 17 sentences accompanied with 3 ground-truth summary sentences. We removed sentences that are not chosen by either model, for the sake of simplicity and readability. The blue sentences show the ones used to generate the ground-truth summary and the green sentences show the sentences that are identified correctly by the extractor w.r.t. the ground-truth summary. Aside from the better coverage that our model offers in finding the relevant sentences for generating the summary, the models also differ in the abstractive summaries they have generated.

As shown in the table, most of the abstraction in the ground-truth summaries happens by removing the extra information that exists in each sentence. The underlined parts in the article represent the part of the sentence that are kept by the abstractor in each method. However, there are small parts of each summary that are the result of paraphrasing. In Table 3.6, we show these abstractive summaries with green-yellow boxes.

According to this example, both perform well in removing extra information from the sentences. Although Fast-RL achieves a good summarization by removing extra information in a sentence, it fails to successfully paraphrase these sentences. This is shown by the text represented by red box in Table 3.6 which shows an incomplete sentence generated by this method. Our model, on the other hand, (shown with the green box) generates a complete sentence.

3.3.3 Analysis of novel n-grams

To analyze the effectiveness of our abstractor module in generating new words, we run another experiment which aims at evaluating the number of novel n-grams generated by the model. For this experiment, we define the novelty score of a model as given in [52]. Let us assume that $ng_n(.)$ captures the unique n-grams in a sentence. Therefore, the novelty score can be defined as:

$$NoveltyScore_n(Y, \hat{Y}) = \frac{|ng_n(\hat{Y}) - ng_n(Y)|}{|ng_n(\hat{Y})|} \tag{3.15}$$

where $|.|$ returns the number of items in a set. Table 3.7 compares the novelty score of our proposed model with the ground-truth and Fast-RL model.

For calculating the ground-truth novelty score, we used the strategy that is explained in Section 3.2.1 to find the closest sentence in the original document and calculated the novelty score using these sentences. As shown in this table, our proposed method, generates up to 6%

[6] sample #10844 in our test dataset.

Table 3.7: Comparison of novelty scores on CNN/DM dataset.

	Novelty Scores				
n	**1**	**2**	**3**	**4**	**5**
Fast-RL (ext+abs+RL) [21]	14.86	17.66	19.3	20.39	21.8
Ext + Abs + GAE (n=2)	**18.65**	**22.18**	**25.48**	**25.48**	**26.4**
Ground-Truth	32.54	46.72	55.42	61.42	65.89

Table 3.8: ROUGE Recall score of our best performing model at 75 bytes on DUC 2004 dataset. For the model shown with $\star$, we used the best performing model shared by the author to get the results on this dataset.

	R-1	**R-2**	**R-L**
ABS+ [101]	28.18	8.49	23.81
RAS-Elman [24]	28.97	8.26	24.06
words-lvt5k-1sent [79]	28.61	9.42	25.24
SEASS [138]	29.21	9.56	25.51
Seq2seq + Selective + MTL + ERAM [58]	29.33	10.24	25.24
Fast-RL (Ext + Abs + RL)* [21]	29.86	7.8	28.31
DRGD [60]	31.79	**10.75**	27.48
Ext + ABS + GAE (n=1)	**32**	8.26	**30.29**

more unique n-grams than Fast-RL model [21] which is due to the fact that our abstractor model significantly improves the performance of the extractor.

3.3.4 Generalization to Other Datasets

To test the generalization ability of our model, we evaluated the performance of our extractive-abstractive framework on DUC 2004 dataset. Most of the existing works that report their result on this dataset are abstractive in nature, which means that a successful model on this dataset requires to select the best part of the document sentence to achieve a good result on this dataset. Table 3.8 shows the result of our proposed model on this dataset.

As shown in this table, our best performing model beats the SOTA result by DRGD model [60] in terms of ROUGE-1 and ROUGE-L while it is inferior when ROUGE-2 is used. Moreover, it is performing more than 2% better than any of the previous methods in terms of ROUGE-L.

3.3.5 Convergence Analysis

In this section, we discuss the convergence and training time of our proposed model and compare it with some of the previous models. We first discuss the training time required

for each of extractor, abstractor, and the joint module w.r.t. some of the other RL-based models and then describe the convergence pattern of our model and Fast-RL. It is known that models that completely focus on doing abstractive summarization from scratch will require significant amount of training time. The pointer-generator model [105] requires more than 3 days to converge, while this is even worse for the pointer-generator with Policy-Gradient objective function [90]. However, similar to the Fast-RL model, our extractive and abstractive framework will only require a few hours of training while achieving far better results.

Aside from training time, the way that model converges over time is also of importance. A stabilized model will be the direct result of a stabilized RL algorithm used for training the joint model. As mentioned before, the GAE model has the ability to reduce the variance of the advantage function which ultimately gives a better stability to the model training. To compare the stability of our model with Fast-RL [21], we used the best performing model shared by the Fast-RL [7] and plot the validation reward collected by this model and our best performing model. Similar to Fast-RL [21], the training process is stopped once there is no improvement on the validation reward after five attempts. Fig 3.4 shows this comparison. As shown in this figure, both methods are not stable in the beginning of the training, however after 1000 iterations, they both reach to their maximum validation reward. However, once the Fast-RL model achieves to its peak reward, the model's performance goes down (the training ends with a lower reward than the reward it achieved at the start of training), while our best performing model achieves a more stable reward over time and after reaching the peak of the reward, the trend of training is mostly plateau over time.

3.3.6 Vulnerability to Noise

Deep neural networks are widely known to be hugely affected by adding a small amount of noise to the input data. This has been shown by a various researches in image classification and natural language processing. To test the vulnerability of our proposed model with noise, we run several other experiment and compare the result with our baseline method.

Adding noise in text data and specifically in text summarization usually come from two changes in the input data.

1. Shuffling sentences: In this approach, we shuffle the sentences of the input document. The goal is to identify whether the summarization model (specifically the extractor module) is able to classify sentences even after shuffling the order of each sentence. Since most of the important sentences in a news article lie in the first three sentences of the article, it is expected that by shuffling the order of sentences, the result of summarization will be poorer.

[7] https://github.com/chenrocks/fast_abs_rl

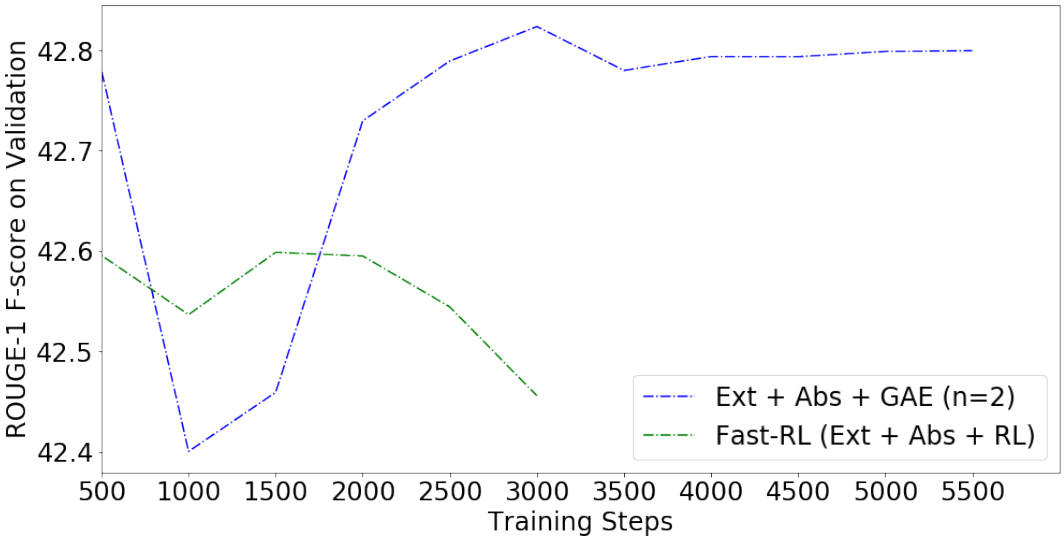


Figure 3.4: The reward achieved on validation dataset during training.

2. Swapping words in a sentence: This test could assess the ability of the abstractor module in rectifying the errors caused by swapping the words in a sentence. We will test to see if the abstractor is able to reconstruct the sentence if a portion of a sentence is swapped. In our experiments, we use the noising ratio of 15% of the length of a sentence to swap words in a sentence.

To test these two scenarios, we designed three different experiments in which in the first one we only shuffle sentences, while in the second scenario, we only swap words in sentences and finally we combine both of these noising techniques to test the resilience of our proposed model. Table 3.9 shows the performance of our proposed model and compare the result with Fast-RL model. Moreover, Table 3.10 shows the performance of the model when we fix a specific sentence and shuffle the rest of sentences in the article.

3.4 Conclusion

In this section, we tackled the problem of abstractive text summarization for news article. Our proposed method comprises of two different modules: (i) An extractor which selects the most salient sentences in the document and (ii) the abstractor which paraphrases these sentences and generates the final summary. Our extractor utilizes a hierarchical temporal CNN-LSTM encoder and an RNN decoder to capture both long-term sentence dependencies in the document and the overall document context to finally select the relevant sentences sequentially. The abstractor, which is paraphrasing model, uses a pointer-generator model

Table 3.9: Vulnerability of our proposed model to noise.

	Noise	R-1	R-2	R-L	Meteor
Ext + GAE (n=2)	1	32.53	11.24	29.43	**18.39**
Ext + ABS + GAE (n=2)	1	**35.59**	**13.03**	**33.27**	16.74
Fast-RL + Ext	1	32.49	11.35	29.40	17.56
Fast-RL + Ext + ABS	1	34.19	11.73	31.8	16.56
Ext + GAE (n=2)	2	38.98	18.29	35.99	**24.69**
Ext + ABS + GAE (n=2)	2	**41.51**	**18.32**	**39.01**	21.46
Fast-RL + Ext	2	39.24	17.94	35.96	24.08
Fast-RL + Ext + ABS	2	40.35	17.88	37.84	21.13
Ext + GAE (n=2)	1 & 2	32.6	11.42	29.68	**18.44**
Ext + ABS + GAE (n=2)	1 & 2	**35.58**	**13.01**	**33.27**	16.76
Fast-RL + Ext	1 & 2	32.48	11.28	29.36	18.34
Fast-RL + Ext + ABS	1 & 2	34.18	11.72	31.79	16.6

Table 3.10: Vulnerability of our proposed model to noise when we fix a sentence in the article.

	Fixed Sent	R-1	R-2	R-L	Meteor
Ext + ABS + GAE (n=2)	1	35.83	13.34	33.43	16.9
Ext + ABS + GAE (n=2)	2	35.71	13.21	33.35	16.83
Ext + ABS + GAE (n=2)	3	35.62	13.12	33.29	16.79

which is trained using Self-Critic Policy-Gradient to provide paraphrases of the sentences selected by the extractor. Finally, we also proposed a co-training framework for bridging the non-differentiable operation of connecting the extractor and abstractor using reinforcement learning. Our co-training model is based on an Actor-Critic framework in which the actor is our extractor-abstractor module and the critic is a regression model trained along with the actor using Generalized Advantage Estimation method. We evaluated our proposed model on non-anonymized CNN/DM dataset and DUC 2004 dataset and demonstrated that our proposed model outperforms various state-of-the-art models on both datasets while reaching a more stable result compared to previous models.

Chapter 4

Transfer Learning in News Article Summarization

As discussed in Chapter 3, in recent years, researchers have used news article datasets e.g., from CNN/DM [40] and Newsroom [37] as a main resource for building and evaluating text summarization models. However, all these models suffer from a critical problem: *a model trained on a specific dataset works well only on that dataset.* For instance, if a model is trained on the CNN/DM dataset and tested on the Newsroom dataset, the result is much poorer than when it is trained directly on the Newsroom dataset. This lack of generalization ability for current state-of-the-art models is the main motivation for our work.

This problem arises in situations where there is a need to perform summarization on a specific dataset, but either no ground-truth summaries exist for this dataset or where collecting ground-truth summaries could be expensive and time-consuming. Thus, the only recourse in such a situation would be to simply apply a pre-trained summarization model to generate summaries for this data. However, as discussed in this chapter, this approach will fail to satisfy the basic requirements of this task and thus fails to generate high quality summaries. Throughout our analysis, we work with two of the well-known news-related datasets for text summarization and one could expect that a model trained on either one of the datasets should perform well on the other or any news-related dataset. On the contrary, as shown in Table 4.1, the Fast-RL model [21] trained on CNN/DM, which holds the state-of-the-art result for text summarization task on the CNN/DM test dataset with 41.18% a F-score according to the ROUGE-1 measure, will reach only a 21.93% on this metric on the Newsroom test data, a performance fall of almost 20%. This observation shows that these models suffer from poor generalization capability.

In this chapter, we first study the extent to which the current state-of-the-art models are vulnerable in generalizing to other datasets and discuss how transfer learning could help in alleviating some of these problems. In addition, we propose a solution based on reinforcement learning which achieves good generalization performance on a variety of summarization datasets. Traditional transfer learning usually works by pre-training a model using a large dataset, fine-tuning it on a target dataset, and then testing the result on that target dataset. However, our proposed method, as shown in Fig 4.1, is able to achieve good results on a variety of datasets by only fine-tuning the model on a single dataset. Therefore, it removes the requirement of training separate transfer models for each dataset. To the best

Table 4.1: The Pointer-Generator [105] and Fast-RL [21] models are trained using the CNN/DM dataset and tested on the CNN/DM and Newsroom datasets.

	Pointer Generator [105]			**Fast-RL** [21]		
ROUGE	1	2	L	1	2	L
CNN/DM	39.77	17.15	36.18	41.18	18.19	38.78
Newsroom	26.59	14.09	23.44	21.93	9.37	19.61

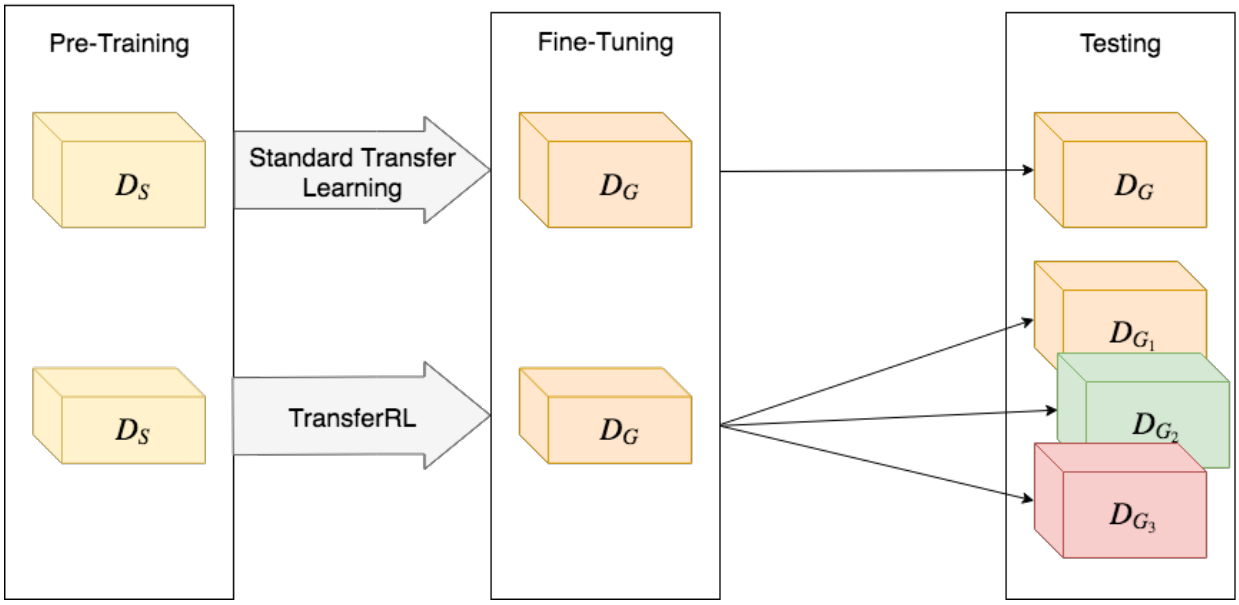


Figure 4.1: In standard transfer learning settings, a model is pre-trained on D_S, all network layers are transferred, the model is fine-tuned using D_G, and finally tested (only) on D_G. On the contrary, our proposed method (TransferRL) uses D_G to create a model that works well on a variety of (test) datasets.

of our knowledge, this is the first work that studies transfer learning for the problem of text summarization and provides a solution for rectifying the generalization issue that arises in current state-of-the-art summarization models. In addition, we conduct various experiments to demonstrate the ability of our proposed method to obtain state-of-the-art results on datasets with small amounts of ground-truth data.

4.1 Related Work

Recently, there has been a surge in the development of deep learning based methods for building models that have the ability to transfer and generalize to other similar problems. Transfer learning (TL) has been well-studied in the domain of image processing; however its utility in NLP problems is yet to be thoroughly investigated. In this section, we will review some of these works.

4.1.1 Transferring Trained Models.

In these works, the underlying model is first trained on a specific dataset and then used as a pre-trained model for another problem or dataset. In this method, depending on the underlying model, one can transform different types of neural network layers from the pre-trained model to the transfer model. Examples of these transferable layers are the word embedding layer, the convolutional layers in a CNN model, Fully Connected (FC) hidden layers, and finally the output layer [93]. Yosinski *et al.* [131] studied the effect of transferring different layers of a deep neural network and found that lower-level layers learn general features while higher level layers capture mostly the specific characteristic of the problem at hand. Researchers have also demonstrated how one can transfer both low-level and high-level neural layers from a CNN for TL [26].

Recently, Semwal *et al.* [106] used this idea of transferring various network layers for text classification. Aside from transferring the network layers, they also experimented with freezing or fine-tuning these layers after the transfer and concluded that fine-tuning the transfer layers will always provide a better result. Moreover, TL has also been studied in the context of the named entity recognition problem [65, 102]. Our proposed method falls into this category. We not only study the effect of transferring network layers, but also propose a new co-training model for training text summarization models using reinforcement learning techniques.

4.1.2 Knowledge Distillation.

Knowledge distillation refers to a class of techniques that trains a small network by transferring knowledge from a larger network. These techniques are typically used when we require building models for devices with limited computational resources [3]. Usually, in these models, there is a teacher (larger model), a student (smaller model), and the goal is to transfer knowledge from teacher to student. Recently, researchers have also used this idea to create models using meta-learning [103], few-shot learning [95, 108], one-shot learning [15, 29], and domain adaptation [34, 116], mostly for image classification problems. However, the effect of these types of models on NLP tasks is yet to be studied or not well studied.

4.1.3 Building Generalized Models.

Recently, McCann *et al.* [71] released a challenge called Decathlon NLP which aims at solving ten different NLP problems with a single unified model. The main intuition behind this model is to comprehend the impact of transferring knowledge from different NLP tasks on building a generalized model that works well on every task. Although this model outperforms some of the state-of-the-art models in specific tasks, it fails to even reach baseline results in tasks like text summarization. We also observe such poor results from other generalized frameworks

such as Google's Tensor2Tensor framework [118].

4.1.4 Text Summarization.

There is a vast amount of research work on the topic of text summarization using deep neural networks [107]. These works range from fully extractive methods [21, 82, 139] to completely abstractive ones [36, 52, 105]. As one of the earliest works on using neural networks for extractive summarization, Nallapati *et al.* [80] proposed a framework that used a ranking technique to extract the most salient sentence in the input. On the other hand, for abstractive summarization, it was Rush *et al.* [101] that for the first time used attention over a sequence-to-sequence (seq2seq) model for the problem of headline generation. To further improve the performance of these models, the pointer-generator model [79] was proposed for successfully handling Out-of-Vocabulary (OOV) words. This model was later improved by using the coverage mechanism [105]. However, all these models suffer from a common problem known as *exposure bias* which refers to the fact that, during training, the model is trained by feeding ground-truth input at each decoder step, while during the test, it should rely on its own output to generate the next token. Also, the training is typically done using cross-entropy loss, while during test, metrics such as ROUGE [66] or BLEU [87] are used to evaluate the model. To tackle this problem, researchers suggested various models using scheduled sampling [10] and reinforcement learning based approaches [21, 90].

Recently, several authors have investigated methods which try to first perform extractive summarization by selecting the most salient sentences within a document using a classifier and then apply a language model or a paraphrasing model [63] on these selected sentences to obtain the final abstractive summarization [21, 82, 139]. However, none of these models, as discussed in this chapter (and shown in Table 4.1) have the capability to generalize to other datasets and thus only perform well for the specific dataset used as target data during the pre-training process.

4.2 Proposed Model

In this chapter, we propose various transfer learning methods for the problem of text summarization. For all experiments, we consider two different datasets: D_S, a *source dataset* used to train the pre-trained model, while D_G, the *target dataset*[1], is the dataset used to fine-tune our pre-trained model. Following the idea of transferring layers of a pre-trained model, our first proposed model transfers different layers of a pre-trained model trained using D_S and fine-tunes them using D_G. We then propose another method which uses a novel reinforcement learning (RL) framework to train the transfer model using training signals received from both D_S and D_G.

[1]Note that, we use D_G instead of D_T for the target dataset to avoid confusing this T subscript with time.

4.2.1 Transferring Network Layers.

There are various network layers used in a deep neural network. For instance, if the model has a CNN encoder and a LSTM decoder, the CNN layers and the hidden decoder layers trained on D_S could be used to fine-tune using D_G. Moreover, the word embedding representation is a key layer in such a model. Either, we use pre-trained word embeddings such as Glove [91] during the training of D_S or let D_S drive the inference of its own word embeddings. We can still let the model to fine-tune a pre-trained word embedding during the training of such a model. In summary, we can transfer the embedding layer, convolutional layer (if using CNN), hidden layers (if using LSTM), and the output layer in a text summarization transfer learning problem. One way to understand the effect of each of these layers is to fine-tune or freeze these layers during model transfer and report the best performing model. However, as suggested by [106], the best performance is realized when all layers of a pre-trained model on D_S are transferred and the model is led to fine-tune itself using D_G. Therefore, we follow the same practice and let the transferred model fine-tune all trainable variables in our model. As shown later in the experimental result section, this way of transferring network layers provides a strong baseline in text summarization and the performance of our proposed reinforced model is close to this baseline. However, one of the main problems with this approach is that, not only should source dataset D_S should contain a large number of training samples but D_G must also have a lot of training samples to be able to fine-tune the pre-trained model and generalize the distribution of the pre-trained model parameters. Therefore, a successful transfer learning using this method requires a large number of samples both for D_S and D_G. This could be problematic, specifically for cases where the target dataset is small and fine-tuning a model will cause over-fitting. For these reasons, we will propose a model which uses reinforcement learning to fine-tune the model only based on the reward that is obtained over the target dataset.

4.2.2 Transfer Reinforcement Learning (TransferRL).

In this section, we explain our proposed reinforcement learning based framework for knowledge transfer in text summarization. The basic underlying summarization mechanism used in our work is the pointer-generator model [105].

Why Pointer-Generator?

The reason we choose a pointer-generator model as the basis of our framework is its ability to handle Out-of-Vocabulary (OOV) words which is necessary for transfer learning. Note that once a specific vocabulary generated from D_S is used to train the pre-trained model, we cannot use a different vocabulary set during the fine-tuning stage on D_G, since the indexing

of words could change for words in the second dataset[2]. According to our experiments, amongst the top 50K words in the CNN/DM and Newsroom datasets, only 39K words are common between the two datasets and thus a model trained on each of these datasets will have more than 11K OOVs during the fine-tuning step. Therefore, a framework that is not able to handle these OOV words elegantly will demonstrate significantly poor results after the transfer. One naïve approach in resolving this problem could be to use a shared set of vocabulary words between D_S and D_G. However, such a model will still suffer from inability to generalize to other datasets with a different vocabulary set.

Pointer-Generator.

As shown in Fig 4.2, a pointer-generator model comprises of a series of LSTM encoders (blue boxes) and LSTM decoders (green boxes). Let us consider dataset $D = \{d_1, \cdots, d_N\}$ as a dataset that contains N documents along with their summaries. Each document is represented by a series of T_e words, i.e. $d_i = \{x_1, \cdots, x_{T_e}\}$, where $x_t \in V = \{1, \cdots, |V|\}$. Each encoder takes the embedding of word x_t as the input and generates the output state h_t. The decoder, on the other hand, takes the last state from the encoder, i.e., h_{T_e}, and starts generating an output of size $T < T_e$, $\hat{\mathbf{Y}} = \{\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_T\}$, based on the current state of the decoder s_t, and the ground-truth summary word y_t. At each step of decoding j, the attention vector α_j, context vector c_j, and output distribution p_{vocab} can be calculated as follows:

$$\begin{array}{lcl} f_{ij} & = & v_1^T tanh(W_h h_i + W_s s_j + b_1) \\ \alpha_j & = & softmax(f_j) \\ c_j & = & \sum_{i=1}^{T_e} \alpha_{ij} h_i \\ p_{vocab} & = & softmax(v_2(v_3[s_j \oplus c_j] + b_2) + b3) \end{array} \tag{4.1}$$

where $v_{1,2,3}$, $b_{1,2,3}$, W_h, and W_s are trainable model parameters and $\oplus$ is the concatenation operator. In a simple sequence-to-sequence model with attention, we use p_{vocab} to calculate the cross-entropy loss. However, since p_{vocab} only captures the distribution of words within the vocabulary, this will generate a lot of OOV words during the decoding step. A pointer-generator model mitigates this issue by using a switching mechanism which either chooses a word from the vocabulary with a certain probability σ or from the original document using the attention distribution with a probability of (1-σ) as follows:

$$\begin{array}{lcl} \sigma_j & = & (W_c c_j + W_s s_j + W_x x_j + b_4) \\ p_j^* & = & \sigma_j p_{vocab} + (1 - \sigma_j) \sum_{i=1}^{T_e} \alpha_{ij} \end{array} \tag{4.2}$$

where W_c, W_x, and b_4 are trainable model parameters and if a word x_j is OOV, then $p_{vocab} = 0$ and the model will rely on the attention values to select the right token. Once the final

[2]For instance, the word "is" could have index 1 in the first dataset while it could have index 10 in the second dataset.

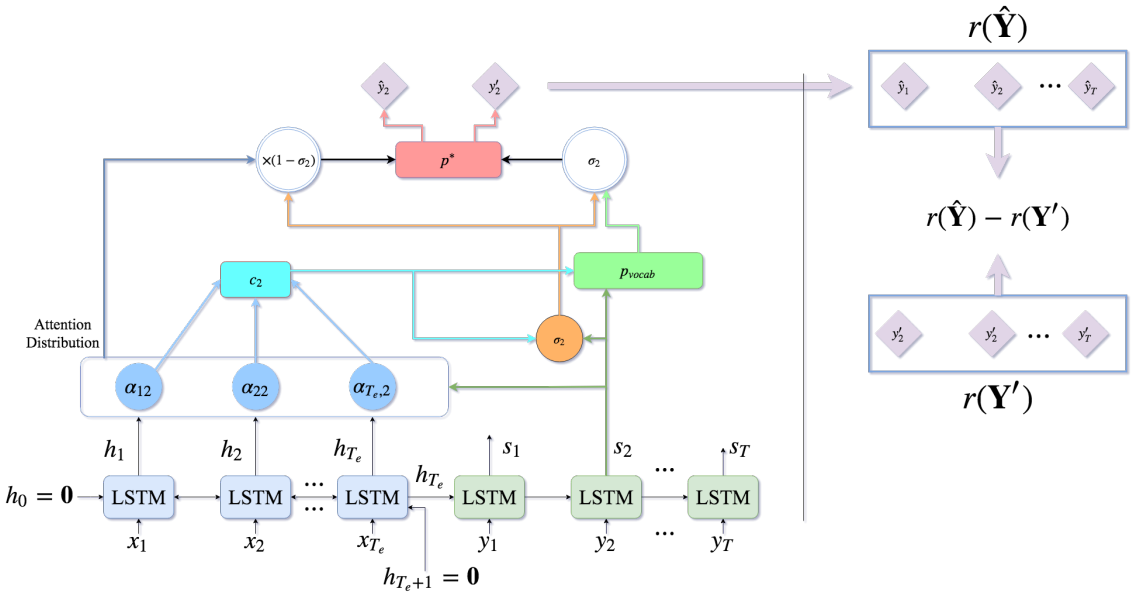


Figure 4.2: Pointer-generator w. self-critic policy gradient

probability is calculated using Eq. (4.2), the cross-entropy (CE) loss is calculated as follows:

$$\mathcal{L}_{CE} = -\sum_{t=1}^{T} \log p_\theta^*(y_t|e(y_{t-1}), s_t, c_{t-1}, \mathbf{X}) \tag{4.3}$$

where θ shows the training parameters and $e(.)$ returns the word embedding of a specific token. However, as mentioned in Section 4.1, one of the main problems with cross-entropy loss is the exposure bias [10, 90] which occurs due to the inconsistency between the decoder input during training and test. A model that is trained using only CE loss does not have the generalization power required for transfer learning, since such a model is not trained to generate samples from its own distribution and heavily relies on the ground-truth input. Thus, if the distribution of input data changes (which can likely happen during transfer learning on another dataset), the trained model will have to essentially re-calibrate every transferred layer to achieve a good result on the target dataset. To avoid these problems, we propose a reinforcement learning framework which slowly removes the dependency of model training on the CE loss and increases the reliance of the model on its own output.

Reinforcement Learning Objective.

In RL training, the focus is on minimizing the negative expected reward rather than on directly minimizing the CE loss. This allows the framework to not only use the model's output for training itself, but also helps in training the model based on the metric that is

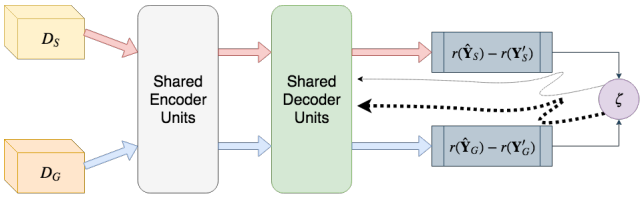


Figure 4.3: The proposed TransferRL framework. The encoder and decoder units are shared between the source (D_S) and target datasets (D_G).

used during decoding (such as ROUGE). To achieve this, during RL training, the following objective is minimized:

$$\text{minimize}\, \mathcal{L}_{RL} = -\mathbb{E}_{y'_1,\cdots,y'_T \sim p^*_\theta(y'_1,\cdots,y'_T)}[r(y'_1,\cdots,y'_T)] \tag{4.4}$$

where $y'_1,\cdots,y'_T$ are sample tokens drawn from the output of the policy (p_θ), i.e., $p^*_1,\cdots,p^*_T$. In practice, we usually sample only one sequence of tokens to calculate this expectation. Hence, the derivative of the above loss function is given as follows:

$$\nabla_\theta \mathcal{L}_{RL} = -\mathop{\mathbb{E}}_{y'_1,\cdots,y'_T \sim p^*_\theta}[\nabla_\theta \log p^*_\theta(y'_1,\cdots,y'_T) r(y'_1,\cdots,y'_T)] \tag{4.5}$$

This minimization can be further improved by adding a baseline reward. In text summarization, the baseline reward could either come from a separate network called a critic network [21] or it could be the reward from a sequence coming from greedy selection over p^*_t [90]. In this work, we consider the greedy sequence as the baseline. In summary, the objective that we minimize during RL training is as follows:

$$\mathcal{L}_{RL} = \sum_t -\log p^*_\theta(y_t|y'_{t-1}, s_t, c_{t-1}, \mathbf{X}) \times \Big(r(\hat{y}_1,\cdots,\hat{y}_T) - r(y'_1,\cdots,y'_T)\Big) \tag{4.6}$$

where $\hat{y}_t$ represents the greedy selection at time t. This model is also known as a *self-critic policy gradient* approach since the model uses its own greedy output to create the baseline. Moreover, the model uses the sampled sentence as the target for training rather than the ground-truth sentence. Therefore, given this objective, the model focuses on samples that do better than greedy selection during training while penalizing those which do worse than greedy selection.

Transfer Reinforcement Learning.

Although a model trained using Eq. (4.6) does not suffer from exposure bias, it can still perform poorly in a transfer learning setting. This is mostly due to the fact that the model is still being trained using the distribution from the source dataset and once transferred to the target dataset, aims to generate samples according to the distribution of the source dataset. Therefore, we need a model that not only remembers the distribution of the source

dataset but also tries to learn and adapt to the distribution of the target dataset. The overall RL-based framework that is being proposed in this chapter is shown in Fig. 4.3. At each step, we select a random mini-batch from D_S and D_G and feed them to the shared encoder units and the decoder starts decoding for each mini-batch. Once the decoding is completed, the model generates a sentence based on greedy selection and another by sampling from the output distribution. Finally, we calculate the TransferRL loss according to Eq. (4.7) and back-propagate the error according to the trade-off parameter ζ. The thick and thin dashed lines in this plot shows the effect of ζ on the extent to which the model needs to rely on either D_S or D_G for back-propagating the error.

Let us consider sequences drawn from greedy selection and sampling from the source dataset D_S and the target dataset D_G as $\hat{\mathbf{Y}}_S$, $\mathbf{Y}'_S$, $\hat{\mathbf{Y}}_G$, and $\mathbf{Y}'_G$, respectively. We will define the transfer loss function using these variables as follows:

$$\mathcal{L}_{TRL} = -\sum_{t=1}^{T} \Bigg((1-\zeta) \log p^*_\theta(y^S_t|\mathbf{U}_S) \Big(r(\hat{\mathbf{Y}}_S) - r(\mathbf{Y}'_S) \Big) + \zeta \log p^*_\theta(y^G_t|\mathbf{U}_G) \Big(r(\hat{\mathbf{Y}}_G) - r(\mathbf{Y}'_G) \Big) \Bigg) \tag{4.7}$$

where $\mathbf{U}_S = \{e_S(y'_{S,t-1}), s_t, c_{t-1}, \mathbf{X}_S\}$, $\mathbf{U}_G = \{e_S(y'_{G,t-1}), s_t, c_{t-1}, \mathbf{X}_G\}$, and $\zeta \in [0, 1]$ controls the trade-off between self-critic loss of the samples drawn from the source dataset and from the target dataset. Therefore, a $\zeta = 0$ means that we train the model only using the samples from the source dataset, while $\zeta = 1$ means that model is trained only using samples from the target dataset. As seen in Eq. (4.7), the decoder state s_t and the context vector c_{t-1} are shared between the source and target samples. Moreover, we use a shared embedding trained on the source dataset, $e_S(.)$ for both datasets while the input data given to the encoder, i.e., X_S and X_G, come from source and target datasets.

In practice, the RL objective loss only activates after a good pre-trained model is obtained. We follow the same practice and first pre-train the model using the source dataset and then activate the transfer RL loss in Eq. (4.7) by combining this loss with the CE loss from Eq. (4.3) using the parameter $\eta \in [0, 1]$ as follows:

$$\mathcal{L}_{Mixed} = (1-\eta)\mathcal{L}_{CE} + \eta\mathcal{L}_{TRL} \tag{4.8}$$

4.3 Experimental Results

We performed a range of experiments to understand the dynamics of transfer learning and to investigate best practices for obtaining a generalized model for text summarization. In this section, we discuss some of the insights we gained through our experiments. All evaluations are done using ROUGE 1, 2, and L F-scores on the test data. All our ROUGE scores have a 95% confidence interval of ± 0.25 as reported by the official ROUGE script[3]. Similar to

[3] https://pypi.org/project/pyrouge/

Table 4.2: Basic statistics for the datasets used in our experiments.

	Newsroom	CNN/DM	DUC'03	DUC'04
# Train	994,001	287,226	0	0
# Val	108,312	13,368	0	0
# Test	108,655	11,490	624	500
Avg. # summary sentences	1.42	3.78	4	4
Avg. # words in summary	21	14.6	11.03	11.43

multi-task learning frameworks such as DecaNLP [71], we use a measure for comparing the result of transfer learning on various datasets by taking the average score of each measure over these datasets. In addition, we also introduce a weighted average score which takes into account the size of each dataset as the weight for averaging the values.

4.3.1 Datasets

We use four widely used datasets in text summarization for our experiments. The first two datasets are Newsroom [37] and CNN/Daily Mail [40] which are used for training our models, while the DUC 2003 and DUC 2004 datasets are only used to test the generalization capability of each model. Table 4.2 shows some of the basic statistics of these datasets. In all these datasets, a news article is accompanied by 1 to 4 human-written summaries and, therefore, will cover a wide range of challenges for transfer learning. For instance, a model that is trained on the Newsroom dataset will most likely generate only one long summary sentence, while for the CNN/DM dataset, the model is required to generate up to four smaller summary sentences. For all experiments, we either use Newsroom as D_S and CNN/DM as D_G, or vice-versa.

4.3.2 Training Setup

For each experiment, we run our model for 15 epochs during pre-training and 10 epochs during the transfer process, and an extra 2 epochs for the coverage mechanism. We use a batch size of 48 during training, the encoder reads the first 400 words, and the decoder generates a summary with 100 words. Both encoder and decoder units have a hidden size of 256 while the embedding dimension is set to 128 and we learn the word embedding during training. For all models, we used the top 50K words in each dataset as the vocabulary and during test we use beam search of size 4. We use AdaGrad to optimize all models with an

Table 4.3: Results on Newsroom, CNN/DM, DUC'03, and DUC'04 test data. D_S shows the dataset that is used during pre-training and D_G is our target dataset. **N** stands for Newsroom and **C** stands for CNN/DM dataset. The method column shows whether we use CE loss, transferring layers (TL), or TransferRL (TRL) loss during training. For each experiment, we run two different setups, with coverage mechanism and without it. This is represented as We use coverage mechanism for all experiments. The result from the proposed method is shown with $\star$.

D_M	D_G	Method	Cov	Newsroom			CNN/DM			DUC'03			DUC'04		
				R_1	R_2	R_L	R_1	R_2	R_L	R_1	R_2	R_L	R_1	R_2	R_L
C+N		CE	No	31.22	18.69	27.94	36.81	15.44	33.18	27.58	8.66	24.61	28.34	8.86	25.54
C+N		CE	Yes	30.26	17.68	27.03	38.23	16.31	34.66	26.71	7.81	24.13	27.96	8.25	25.25
N		CE	No	36.08	24.23	32.79	33.67	12.79	29.78	28.22	9.53	24.99	30.19	10.44	27.06
N		CE	Yes	36.16	24.33	32.87	33.58	12.76	29.72	28.03	9.15	24.75	29.85	10.3	26.7
N	**C**	TL	No	31.43	19.07	28.13	36.45	14.87	32.66	27.69	8.64	24.54	29.01	9.19	26.14
N	**C**	TL	Yes	35.37	23.45	32.07	34.51	13.49	30.61	28.19	9.34	24.96	29.83	9.98	26.66
N	**C**	TRL*	No	36.04	24.22	32.78	33.65	12.76	29.77	28.32	9.5	25.12	30.37	10.58	27.28
N	**C**	TRL*	Yes	**36.5**	**24.77**	**33.25**	35.24	13.56	31.33	**28.46**	**9.65**	**25.45**	**30.45**	**10.63**	**27.42**
C		CE	No	26.4	13.25	23.07	39.11	16.81	35.64	25.62	6.68	23.09	26.25	6.97	23.76
C		CE	Yes	26.59	14.09	23.44	39.77	17.15	36.18	25.66	6.86	23.16	27.12	7.07	24.57
C	**N**	TL	No	35.85	24.06	32.56	34.39	13.29	30.49	28.4	9.63	25.37	30.13	10.38	27.06
C	**N**	TL	Yes	35.39	23.69	32.13	35.31	13.88	31.4	28.34	9.33	25.09	30.05	10.1	26.96
C	**N**	TRL*	No	26.6	13.41	23.15	39.18	17.01	35.72	25.57	6.7	23.05	26.31	6.98	23.85
C	**N**	TRL*	Yes	27.34	14.23	24.01	**39.81**	**17.23**	**36.31**	25.61	6.78	23.18	27.14	7.11	24.71

initial learning rate of $\gamma_0 = 0.15$ during pre-training and $\gamma_0 = 0.001$ during RL and coverage and linearly decrease this learning rate based on the epoch numbers as $\gamma_t = \gamma_0/epoch$. Moreover, ζ is set to zero at the start of RL training and is increased linearly so that it gets to 1 by the end of training. During RL training, we use scheduled sampling with sampling probability equal to the ζ value. We use the RLSeq2Seq [47] framework to build our model.

4.3.3 Effect of Dataset Size

We will now discuss some of the insights we gained starting with understanding the effect of data size for pre-training. According to our experiments (as shown in Table 4.4), on average, a model trained using the Newsroom dataset as the source dataset D_S has much better performance than models that use CNN/DM as the D_S in almost all configurations. This is not a surprising result since deep neural networks are data hungry models and typically work the best when provided with a large number of samples. The first experiment in Table 4.3 and Table 4.4 uses only the Newsroom dataset for training the model and not surprisingly it performs good on this dataset; however as discussed earlier, its performance on other datasets is poor.

4.3.4 Common Vocabulary

As mentioned in Section 4.2, one way to avoid excessive OOV words in transfer learning between two datasets is to use a common vocabulary between D_S and D_G and train a model using this common vocabulary set. Although a model trained using such a vocabulary could perform well on these two datasets, it still suffers from poor generalization to other unseen datasets. To demonstrate this, we combine all articles in CNN/DM and Newsroom

Table 4.4: Normalized and weighted normalized F-Scores for Table 4.3.

D_S	D_G	Method	Cov	Avg. Score			Weighted Avg. Score		
				R_1	R_2	R_L	R_1	R_2	R_L
C+N		CE	No	30.99	12.91	27.82	32.47	17.95	29.11
C+N		CE	Yes	30.79	12.51	27.77	32.04	17.36	28.74
N		CE	No	32.04	14.25	28.66	35.53	21.66	32.11
N		CE	Yes	31.91	14.14	28.51	35.58	21.73	32.16
N	**C**	TL	No	31.15	12.94	27.87	32.55	18.12	29.14
N	**C**	TL	Yes	31.98	14.07	28.58	35.17	21.21	31.74
N	**C**	TRL⋆	No	32.1	14.27	28.74	35.5	21.64	32.1
N	**C**	TRL⋆	Yes	**32.66**	**14.65**	**29.36**	**36.21**	**22.25**	**32.81**
C		CE	No	29.35	10.93	26.39	29.25	14.04	25.89
C		CE	Yes	29.79	11.29	26.84	29.54	14.77	26.29
C	**N**	TL	No	32.26	14.34	28.87	35.52	21.64	32.09
C	**N**	TL	Yes	32.27	14.25	28.9	35.37	21.48	31.96
C	**N**	TRL⋆	No	29.42	11.03	26.44	29.42	14.21	25.97
C	**N**	TRL⋆	Yes	29.98	11.34	27.05	30.13	14.9	26.76

training datasets to create a single unified dataset (C+N in Table 4.3 and Table 4.4) and train a model using CE loss in Eq. (4.3) and the common set of vocabulary. The result of this experiment is shown as Experiment 2 in Table 4.3 and in Table 4.4. As shown here, by comparing these results to Experiment 1, we see that combining these two datasets will decrease the performance on Newsroom, DUC'03, and DUC'04 test datasets but will increase the performance for CNN/DM test data. Moreover, by comparing the generalization ability of this method on DUC'03 and DUC'04 datasets, we see that it performs up to 2% worse than the proposed method. This is also witnessed by comparing the average scores and weighted average scores of our proposed model against this model. On average, our method improves up to 4% compared to this method according to the R_1 weighted average score.

4.3.5 Transferring Layers

In this experiment, we discuss the effect of transferring different layers of a pre-trained model for transfer learning. In the pointer-generator model described in Section 4.2, the embedding matrix, the encoder and decoder model parameters are the choices for layers we can use for transfer learning. For this experiment, we pre-train our model using D_S and during transfer learning, we replace D_S with D_G and continue training of the model with CE loss. As shown in Tables 4.3 and 4.4 this way of transferring network layers provides a strong baseline for comparing the performance of our proposed method. These results show that even a simple transferring of layers could provide enough signals for the model to adapt itself to the new

Table 4.5: Result of TransferRL after clipping ζ at 0.5 and $\zeta = 1.0$ on Newsroom, CNN/DM, DUC'03, and DUC'04 test data along with the average and weighted average scores.

D_M	D_G	ζ	Cov	Newsroom			CNN/DM			DUC'03			DUC'04		
				R_1	R_2	R_L	R_1	R_2	R_L	R_1	R_2	R_L	R_1	R_2	R_L
N	**C**	1.0	No	36.04	24.22	32.78	33.65	12.76	29.77	28.32	9.5	25.12	30.37	10.58	27.28
N	**C**	1.0	Yes	**36.5**	**24.77**	**33.25**	35.24	13.56	31.33	**28.46**	**9.65**	**25.45**	**30.45**	**10.63**	**27.42**
C	**N**	1.0	No	26.6	13.41	23.15	39.18	17.01	35.72	25.57	6.7	23.05	26.31	6.98	23.85
C	**N**	1.0	Yes	27.34	14.23	24.01	**39.81**	**17.23**	**36.31**	25.61	6.78	23.18	27.14	7.11	24.71
N	**C**	0.5	No	36.2	24.38	32.92	33.76	12.86	29.87	28.38	9.61	25.12	30.34	10.56	27.17
N	**C**	0.5	Yes	36.06	24.23	32.78	33.7	12.83	29.81	28.3	9.54	25.04	29.88	10.23	26.8
C	**N**	0.5	No	26.4	13.25	23.07	39.1	16.81	35.63	25.58	6.69	23.08	26.22	6.91	23.75
C	**N**	0.5	Yes	27.52	14.46	24.19	38.82	16.64	35.22	24.8	6.49	22.35	25.65	6.72	23.19

data distribution. However, as discussed earlier in Section 4.2, this way of transfer learning tends to completely forget the pre-trained model distribution and entirely changes the final model distribution toward the dataset used for fine-tuning. This effect can be observed in Table 4.3 by comparing the result of experiments 1 and 3. As shown in this table, after transfer learning the performance drops on the Newsroom test dataset (from 36.16 to 35.37 based on R_1) while it increases on the CNN/DM dataset (from 33.58 to 34.51 based on R_1). However, since our proposed method tries to remember the distribution of the pre-trained model (through the ζ parameter) and slowly changes the distribution of the model according to the distribution coming from the target dataset, it performs better than simple transfer learning on these two datasets. This is shown by comparing the result in experiments 3 and 4 in Table 4.3, which shows that our proposed model performs better than naïve transfer learning in all test datasets.

4.3.6 Effect of Zeta

As mentioned in Section 4.2, the trade-off between emphasizing the training to samples drawn from D_S or D_G is controlled by the hyper-parameter ζ. To see the effect of ζ on transfer learning, we clip the ζ value at 0.5 and train a separate model using this objective. Basically, a $\zeta = 0.5$ means that we treat the samples coming from source and target datasets equally during training. Therefore, for these experiments, we start ζ at zero and increase it linearly till the end of training but clip the final ζ value at 0.5. Table 4.5 and 4.6 show the result of this experiment. For simplicity sake, we provide the result of our proposed model achieved from not clipping ζ along with these results. By comparing the results from these two setups, we can see that, on average, increasing the value of ζ to 1.0 will yield better results than clipping this value at 0.5. For instance, according to the average and weighted average score there is an increase of 0.7% in ROUGE-1 and ROUGE-L scores when we do not clip the ζ at 0.5. By comparing the CNN/DM R_1 score in this table, we see that clipping the ζ value will definitely hurt the performance on D_G since the model shows equal attention to the distribution coming from both datasets. On the other hand, the surprising component here is that, by avoiding ζ clipping, the performance on the source dataset also increases [4].

[4]For $\zeta \in (0.5, 1)$, we have only seen small improvement in the results and hence we omitted them from this section.

Table 4.6: Normalized and weighted normalized F-Scores for Table 4.5.

D_S	D_G	ζ	**Cov**	**Avg. Score**			**Weighted Avg. Score**		
				R_1	R_2	R_L	R_1	R_2	R_L
N	**C**	1.0	No	32.1	14.27	28.74	35.5	21.64	32.1
N	**C**	1.0	Yes	**32.66**	**14.65**	**29.36**	**36.21**	**22.25**	**32.81**
C	**N**	1.0	No	29.42	11.03	26.44	29.42	14.21	25.97
C	**N**	1.0	Yes	29.98	11.34	27.05	30.13	14.9	26.76
N	**C**	0.5	No	32.17	14.36	28.77	35.65	21.79	32.23
N	**C**	0.5	Yes	31.99	14.21	28.61	35.53	21.66	32.11
C	**N**	0.5	No	29.33	10.92	26.38	29.24	14.04	25.88
C	**N**	0.5	Yes	29.2	11.08	26.24	30.05	14.94	26.66

4.3.7 Transfer Learning on Small Datasets

As discussed in Section 4, transfer learning is good for situations where the goal is to do summarization on a dataset with little or no ground-truth summaries. For this purpose, we conducted another set of experiments to test our proposed model on transfer learning using DUC'03 and DUC'04 datasets as our target datasets, i.e., D_G. For these experiments, we randomly pick 20% of each dataset as our training set, 10% as a validation dataset, and the rest of the dataset as our test data. This will generate 124 and 100 articles as our training dataset for DUC'03 and DUC'04, respectively. Similar to other experiments in this paper, we use CNN/DM and Newsroom as D_S and use DUC'03 and DUC'04 as D_G during transfer learning. Due to the size of these datasets, the models are trained only for 3000 iterations during fine-tuning and the best model is selected according to the validation set. Tables 4.7 and 4.8 depict the results of this experiment. As shown in these tables, for DUC'03, when we simply transfer network layers, it performs slightly better (not statistically higher according to a 95% confidence interval) than our proposed model; however, our proposed model will achieve a far better result on DUC'04. As shown in these tables, the results achieved from fine-tuning a pre-trained model using these datasets is very close to the ones achieved in Table 4.3 and in the case of the DUC'04 dataset, our proposed method in Table 4.3 achieves even better results than the ones shown in Table 4.8. This shows the ability of our proposed framework in generalizing to unseen datasets. Note that, unlike these experiments, the proposed model in Table 4.3 has no information about the data distribution of DUC'03 and DUC'04 and still performs better on these datasets.

4.3.8 Other Generalized Models

We also compare the performance of our proposed model against some of the recent methods from multi-task learning. In text summarization, the DecaNLP [71] and Tensor2Tensor [118]

Table 4.7: Result of transfer learning methods using Newsroom for pre-training and DUC'03 for fine-tuning. The underlined result shows that the improvement from TL is not statistically significant compared to the proposed model.

D_S	D_G	**Method**	**Cov**	R_1	R_2	R_L
N	**DUC'03**	TL	No	28.76	9.73	25.54
N	**DUC'03**	TL	Yes	<u>28.9</u>	<u>9.67</u>	<u>25.56</u>
N	**DUC'03**	TRL*	No	28.29	9.18	24.91
N	**DUC'03**	TRL*	Yes	<u>28.76</u>	<u>9.5</u>	<u>25.39</u>
C	**DUC'03**	TL	No	27.51	8.17	25.57
C	**DUC'03**	TL	Yes	28.08	7.83	25.32
C	**DUC'03**	TRL*	No	24.94	6.21	22.44
C	**DUC'03**	TRL*	Yes	24.68	6.32	22.15

Table 4.8: Result of transfer learning methods using Newsroom for pre-training and DUC'04 for fine-tuning.

D_S	D_G	**Method**	**Cov**	R_1	R_2	R_L
N	**DUC'04**	TL	No	30.22	10.55	27.15
N	**DUC'04**	TL	Yes	27.68	9.09	25.29
N	**DUC'04**	TRL*	No	**30.35**	**10.43**	**27.32**
N	**DUC'04**	TRL*	Yes	29.54	9.99	26.56
C	**DUC'04**	TL	No	28.45	8.41	26.75
C	**DUC'04**	TL	Yes	28.45	8.05	26.54
C	**DUC'04**	TRL*	No	25.98	6.71	23.51
C	**DUC'04**	TRL*	Yes	26.05	6.63	23.54

Table 4.9: Comparing our best performing model with state-of-the-art multi-task learning frameworks on the CNN/DM dataset and according to the average of ROUGE 1, 2, and L F-scores. The result with $*$ is the same as reported in the original paper.

	DecaNLP [71]	**Tensor2Tensor [118]**	**Proposed Model**
Average ROUGE	25.7*	27.4	**31.12**

are two of the most recent frameworks that use multi-task learning. Following the setup in these works, we focus on models that are trained using CNN/DM datasets and report the average ROUGE 1, 2, and L F-scores for our best performing model. Table 4.9 compares the result of our proposed approach against these methods.

Chapter 5

Text Summarization with Auxiliary Training

In recent years and with the rise of general purpose deep learning frameworks, such as seq2seq models, multi-task learning has shown great promises on improving traditional methods. In **multi-task learning**, we usually combine various different tasks that share a common infrastructure. For instance, we can combine similar tasks such as headline generation, text summarization, and sentence paraphrasing. Although the objective function for each of these methods differ, but the underlying framework that is used to train each of these tasks is usually a seq2seq model with and encoder and decoder. Usually, in multi-task learning, all tasks share a common encoder while the decoder is task specific. During training, we feed a combination of mini-batches containing samples from different tasks and calculate the loss for each respective task and finally aggregate them together to generate the final loss. The goal is to improve the performance of all tasks such that, using this general framework, we achieve better results on each task comparing to the case where we only train a model using a single task.

On the other hand, in **auxiliary-task learning**, although we still use various tasks to train the model, but there is only one primary task that is required to improve. This way, all the other tasks are only helping the primary task to achieve a better result and we do not care for the performance of these auxiliary tasks.

However, there is a problem with current multi-task/auxiliary-task learning and the problem lies mostly in the way the loss from different tasks are aggregated to generate the final loss.

Let us assume that the gradient of loss for each task is shown by ∇L_{F_i} and after each step of training, we will calculate the gradient and use an optimization algorithm to minimize the loss. Let us show the gradient of loss for all task as follows:

$$\nabla \mathcal{L} = \sum_{i=1}^{N} w_i \nabla \mathcal{L}_{F_i} \tag{5.1}$$

where N represents the number of tasks and w_i shows the influence of a specific gradient in calculating the complete gradient. In single task learning, w_i is zero for all tasks except one, while in the multi-task learning, each task has a separate value. Assigning higher value for the primary task will make the optimization to focus on the primary task, while assigning higher values to auxiliary tasks will put the optimization attention on those tasks.

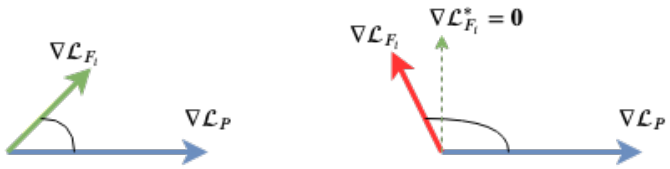


Figure 5.1: The positive and negative transfer. The left figure shows the positive transfer where the gradient of the auxiliary task, e.g. $\mathcal{L}_{F_i}$ is positively correlated with the gradient of primary task, e.g. $\mathcal{L}_P$. While in the right figure, which shows the negative transfer, $\mathcal{L}_{F_i}$ is not following the direction of the primary task.

Our goal is to minimize this loss. However, what would happen if the gradient of some of our tasks do not follow the direction of the gradient of the primary task? or in a more severe situation, what would happen if they completely go on the opposite direction of the primary task? This phenomenon which is represented in Fig 5.1, is called *negative transfer* and has been one of the most important challenges in auxiliary training [28]. As shown in Fig 5.1, each gradient has not only a direction but also a magnitude or weight that will finally lead the final gradient of loss for all tasks to the direction with larger weights. Having this problem, in multi-task learning framework usually will cause a slow convergence and poor result [28]. This phenomenon is also shown in our experiments. Fig 5.3 shows the number of positively and negatively correlated gradient with the primary task. In this experiment, the primary task is extractive summarization while the auxiliary task are sentiment analysis (classification) and scoring a sentence based on its subjectivity (regression). As shown in this figure, the number of parameters suffering from the negative transfer is larger than the positively correlated parameters in this problem.

In this chapter, we will explore how adding auxiliary tasks to text summarization could potentially improve the performance of each sub-module and propose a solution for avoiding the negative transfer in text summarization. To the best of our knowledge this is the first work that uses auxiliary task learning on text summarization and provide a solution for improving the negative transfer in this problem.

Section 5.1 will discuss some of the recent works on auxiliary learning and Section 5.2 provides our proposed method and Section 5.3 shows our experimental results and finally we conclude the chapter in Section 5.4.

5.1 Related Work

Using extra tasks to improve the performance of a primary task or subsequently the performance of those extra tasks is known to provide promising results in different domains [85]. This has been first observed by Caruana *et al.* [18] which were able to improve the result of a survival analysis by introducing auxiliary tasks to their main problem. The researches in this area could be roughly divided into two categories: methods that try to find relatedness of tasks and methods that try to avoid negative transfer irrespective of how related each

task is to each other.

There are a lot of works done on assessing the relatedness of tasks in a multitask learning problem. Rosenstein *et al.* [99] empirically showed that if two tasks are too dissimilar, then brute-force transfer may hurt the performance of the target task. Some works have been utilized task clustering to analyze relatedness among tasks [5, 9] which potentially could provide some insights on how to avoid the negative transfer phenomenon. Eaton *et al.* [31] used a graph-based model in which the relationships between tasks are modeled by embedding the set of learned models in a graph using transferability as the metric. Mahmud *et al.* [68] studied the case of transfer learning using Kolmogorov complexity to measure relatedness between tasks using a Bayesian framework. In a similar work, Bakker*et al.* [5] adopted another Bayesian approach with some shared parameters for all tasks and some loosely connected parameters that are learned from the data. Argyriou *et al.* [2] divided the learning tasks into groups based on low-dimensional representation that they share. Thus, tasks within a group can find it easier to transfer useful knowledge.

In the past few years, multi-task and auxiliary-task learning has been widely used in both NLP and computer vision [72, 77, 114, 120, 136]. The main building block for these problems is to identify a unified framework that could potentially be used by different sub-tasks. As one of the earliest attempt in providing such unified framework, in NLP and image processing, was the Tensor2Tensor [118] framework. Tensor2Tensor provided a single seq2seq framework for various different tasks including but not limited to image classification, image generation, language modeling, sentiment classification, and text summarization. Although this unified framework could potentially be used to train multiple tasks, the Tensor2Tensor did not provide any capability for multi-task learning. A similar framework was later proposed by McCann *et al.* [71] for the multi-task learning problem. In their unified framework, they combined ten different NLP tasks in a single model and was able to improve the performance of some of the sub-tasks.

Although multi-task/auxiliary-task learning could provide SOTA result for some of the sub-tasks but the performance of the overall framework is somewhat limited. This is heavily due to a problem known as *negative transfer* which refers to a phenomenon where the gradient of auxiliary tasks do not follow the direction of the gradient of the primary task [85]. This issue will not only cause a slower convergence for the whole training model, but also could severely affect the optimization of the model. As a solution to this problem, researchers proposed using auxiliary tasks that are well-aligned with primary task [45, 74, 135], however the success of these models are limited since we are only bound to tasks that are similar to the primary task. One way to mitigate the effect of the negative transfer is to project the gradient of the auxiliary tasks to the half-space of vectors whose cosine similarity with the gradient of main task is positive. This idea was first proposed by Du *et al.* [28] where they suggested to use unweighted and weighted cosine projection to mitigate this problem. However, even these projections could some times fail to find the right optimal point. To avoid the problems with these methods, we explore a different gradient projection and show how it could potentially improve the result of the primary task using auxiliary training.

5.2 Proposed Method

Let us assume that our primary task, P is the extractive summarization which is in turn a binary classification. Naturally, we would like to select auxiliary tasks that are relevant to this problem. Let us show these tasks using $F_i, i \in \{1 \cdots N\}$ where N is the number of auxiliary tasks. As we mentioned in Section 5, the difference between multi-task learning and auxiliary training is the way we incorporate the gradient of loss from each auxiliary tasks.

In multi-task learning, given our primary tasks and a set of auxiliary tasks, the gradient of the overall loss is calculated as follows:

$$\nabla\mathcal{L} = \nabla\mathcal{L}_P + \lambda \sum_{i=1}^{N} \nabla\mathcal{L}_{F_i} \tag{5.2}$$

where λ controls the magnitude of auxiliary loss's effect on the overall loss and is a hyper-parameter set by the user.

However, in the auxiliary training, we can control the inclusion and exclusion of a single task during training as follows:

$$\nabla\mathcal{L} = \nabla\mathcal{L}_P + \lambda \sum_{i=1}^{N} \nabla\mathcal{L}^*_{F_i} \tag{5.3}$$

where $\mathcal{L}^*$ in its most simplistic form could be defined as follows:

$$\nabla\mathcal{L}^*_{F_i} = \begin{cases} \nabla\mathcal{L}_{F_i} & \cos(\nabla\mathcal{L}_{F_i}, \nabla\mathcal{L}_P) > 0 \\ 0 & \text{o.w} \end{cases} \tag{5.4}$$

As shown in Eq.(5.4), every time that the gradient of an auxiliary task is not following the direction of the primary task, we replace it with zero and discarding its effect on the final loss. This is a simple but effective way of cancelling out the negative transfer which was first proposed by [28] and is known as *unweighted cosine*. We can also assign weights to the gradient of each auxiliary task by the cosine similarity as follows:

$$\nabla\mathcal{L}^*_{F_i} = max(0, cos(\nabla\mathcal{L}_{F_i}, \nabla\mathcal{L}_P)).\nabla\mathcal{L}_{F_i} \tag{5.5}$$

Another way to handle this issue is by normalizing the gradient of the primary task by its norm and use the dot product of the two gradients as the weight:

$$\nabla\mathcal{L}^*_{F_i} = \nabla\mathcal{L}_{F_i} - min(0, \nabla\mathcal{L}_{F_i}.\frac{\mathcal{L}_P}{||\mathcal{L}_P||}).\frac{\mathcal{L}_P}{||\mathcal{L}_P||}) \tag{5.6}$$

This method is called projection and if $\nabla\mathcal{L}_P$ and $\nabla\mathcal{L}_{F_i}$ form an acute angle both projection and unweighted cosine leave the auxiliary gradient unchanged while with the weighted cosine will reduce its norm based on the similarities between the two vectors. Particularly, the

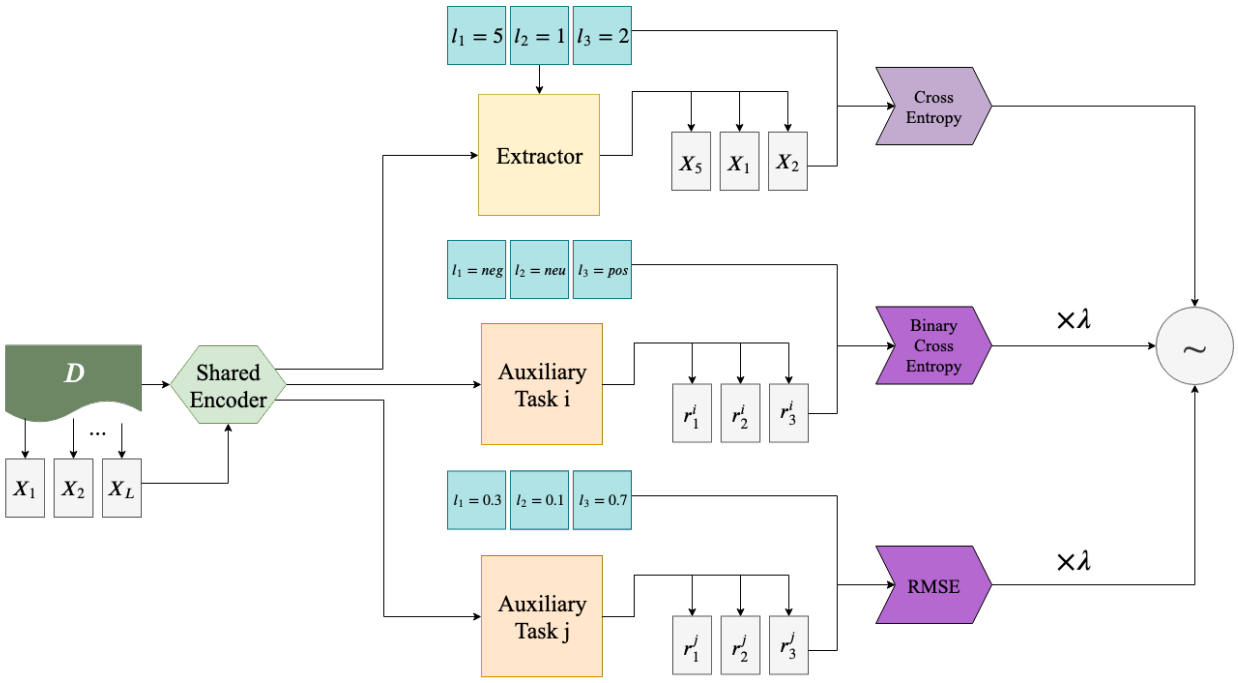


Figure 5.2: The overall framework of our proposed model with two auxiliary tasks. Similar to the primary task, the auxiliary tasks are supervised tasks which receive the same input as the extractor and different ground-truth labels depending on the task. Once each model generated its own output, we can calculate the loss using the respective loss for each task. Finally, the grey circle at the end of the loss calculation show how we combine these losses depending on the approach that we use to mitigate the negative transfer.

weighted cosine strongly shrinks the auxiliary gradient when it becomes almost orthogonal to the primary gradient. However, in case of a negative transfer, both unweighted and weighted cosine simply ignore the auxiliary task's gradient and project will only negate the components of $\nabla\mathcal{L}_{F_i}$ which are colinear with $\nabla\mathcal{L}_P$. We also suggest using another method which as shown in the experiment section achieves a comparably better result than the previous methods. In this method, we use the sign function to decide when to keep and discard the gradient as follows:

$$\nabla\mathcal{L}^*_{F_i} = \nabla\mathcal{L}_{F_i} \times (1 + sign(\nabla\mathcal{L}_{F_i} \times \nabla\mathcal{L}_P))/2 \tag{5.7}$$

This is a similar approach to the unweighted cosine but rather using cosine similarity we use the pair-wise multiplication to decide when to keep and discard the gradients.

In general, unweighted and weighted cosine completely ignore the gradient of an auxiliary task if it does not follow the direction of the primary task, while with projection, it tries to negate the direction of only those elements that are not following the same direction as the primary task and leave others untouched.

Fig 5.2 shows an example of how this method works for the extractive summarization problem.

5.3 Experiments

We use the CNN/DM dataset as the main dataset throughout this section. We consider the extractive summarization as our primary task. In this task, for each sentence, we will classify it into either a salient or non-salient categories. We use the framework proposed in Section 4.2 for our extractor. As mentioned earlier, in multi-task/auxiliary learning, we usually add tasks that are related to the primary task. Therefore, for our experiments, we are also interested in tasks that first work on a sentence level and second have the same nature as a classification problem.

To serve both of these conditions, we are considering the sentiment analysis and subjectivity analysis as our auxiliary task for this dataset. The sentiment analysis task is a multi-class classification, which classifies each sentence into one of the following classes: positive, negative, neutral. On the other hand, the subjectivity analysis is a regression problem, in which for each sentence, we generate a score that shows the subjectivity score of that sentence. We have also done another experiment for classifying each document based on their topic. However, since the topics in CNN/DM dataset are poorly designed and capturing a general topic, e.g. Sport, National, etc, the classification result will perform very close to a random classifier and ultimately it will not improve the result of our primary task.

Since CNN/DM dataset only contains ground-truth summary sentences for each article, we have to generate the ground-truth sentiment and subjectivity scores for all the sentences in the article. We use the Vader sentiment analysis tool [44] to generate the ground-truth sentiments for each sentence in an article and use the TextBlob [1] library to extract the subjectivity scores.

The λ value is set with two extreme values of 1 and 300 and we use the moving average over the gradient of the primary task to update our weights. This will provide a smoother training for our primary objective.

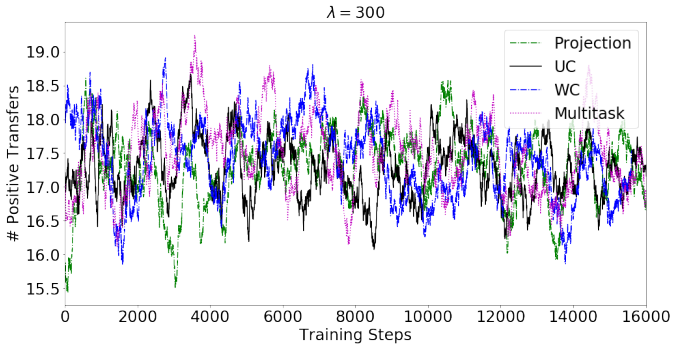
Our experimental section is divided in two parts:

1. we validate that the negative transfer phenomenon actually happens in the extractive summarization task.
2. we also show how this method improves the results of our single-tasked extractive summarization.

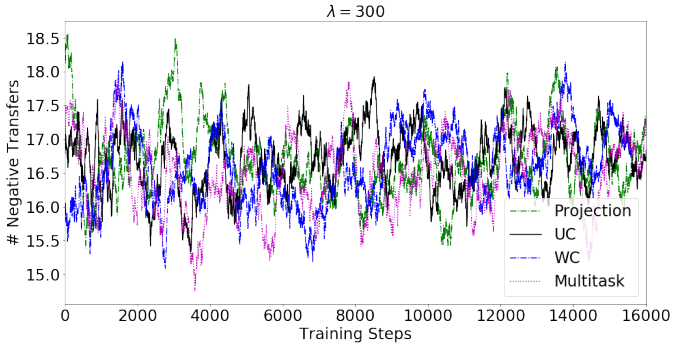
5.3.1 Negative Transfer Occurrence

To show the occurrence of the negative transfer phenomenon, at each step of training, we count the number of parameters that follow the gradient of the primary task (positive trans-

[1] https://textblob.readthedocs.io/en/dev/



(a) # of Positive Transfers



(b) # of Negative Transfers

Figure 5.3: Number of positively and negatively correlated parameters with the primary task.

fer) and do the same for those that negatively correlate with the primary task (negative transfer).

Fig 5.3 shows the changes in the number of positive and negative transfers at each step of the training for the four methods explained in Section 5.2. As shown in this figure, when $\lambda == 300$, the number of negative and positive transfer is always constant. This is counter-intuitive since we might expect that by avoiding the negative transfer, the number of negatively correlated gradient should decrease as we go on with the training. However, as shown in this figure, this does not happen and although the number of positives and negative transfers fluctuates a lot during the training, its rate of occurrences seems to be a constant value. Although this might look like a troubling issue, we should note that the whole purpose of avoiding the negative transfer is to have a better training model for our primary task not to decrease the number of negative transfers in our model.

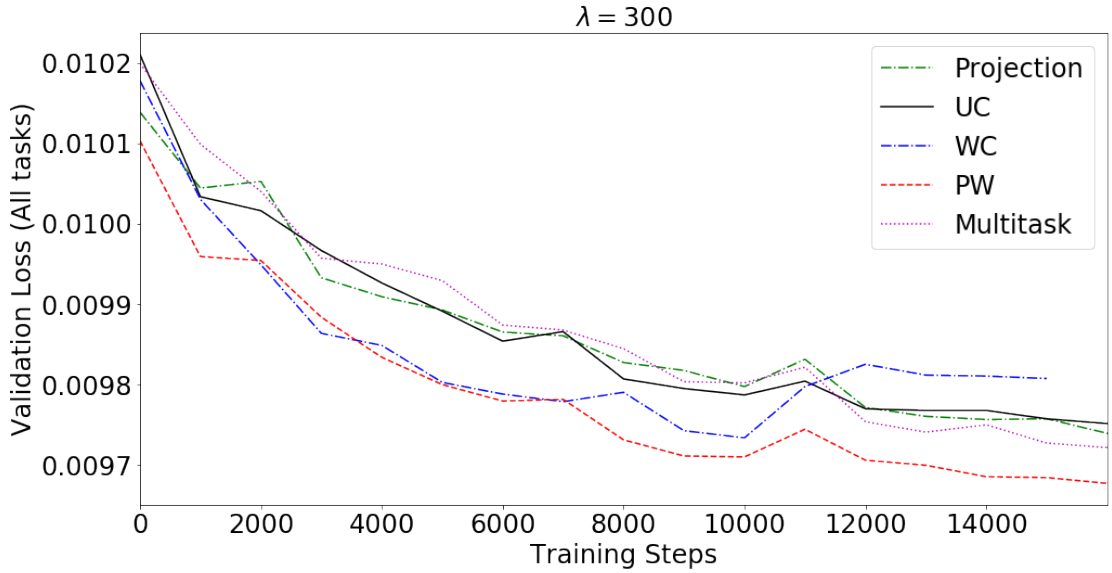


Figure 5.4: Validation Loss for the extractive text summarization during the auxiliary training with each method.

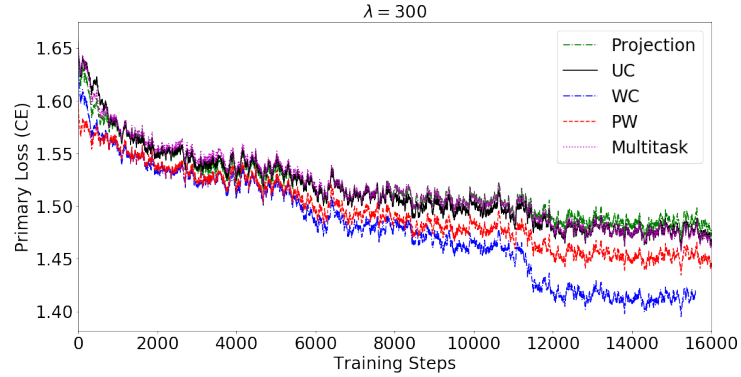
5.3.2 Improving Extractive Summarization Result w. Auxiliary Training

We explore how each of the methods proposed in this chapter will potentially improve or decline the performance of the extractive summarization. We have done our analysis using unweighted cosine (UC), weighted cosine (WC), Projection, pair-wise (PW) methods and we compare them with the single task and multi-task learning according to the speed of convergence, the extractive results on the CNN/DM dataset, and the performance on each auxiliary tasks.

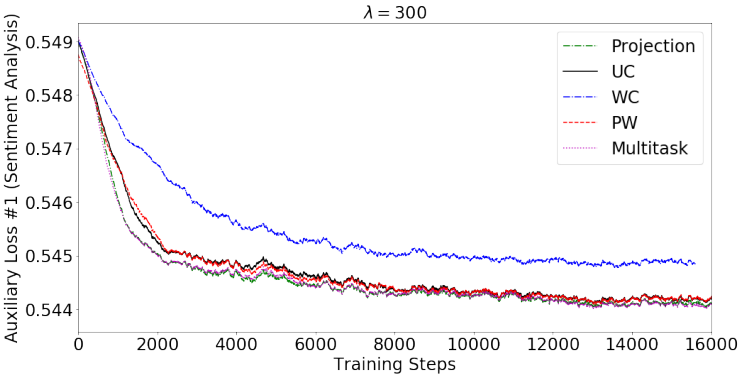
Convergence Analysis

Fig 5.4 shows the validation loss achieved during training of the auxiliary-training using these methods. As shown in this figure, the method based on the Weighted Cosine starts over-fitting on the primary task after a while. This is also confirmed by looking at Fig 5.5a which shows the primary loss during training using these methods. As shown in this figure, WC keeps decreasing the primary loss, however as shown in Fig 5.4, this is the result of over-fitting to the model.

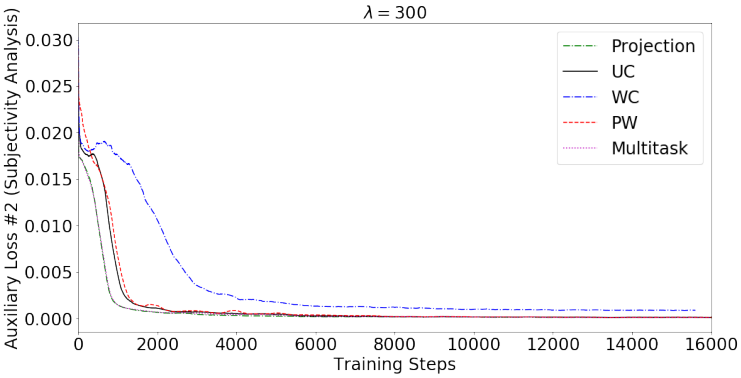
On the other hand, according to the validation loss, the most stable and consistent method is the Pair-Wise method which has a steady performance during the training. One the most important insight regarding all of these models is that they all converge to the best



(a) Primary Loss



(b) Sentiment analysis loss

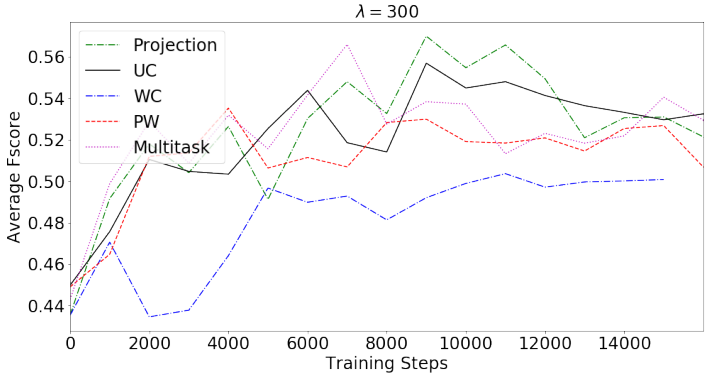


(c) Subjectivity analysis loss

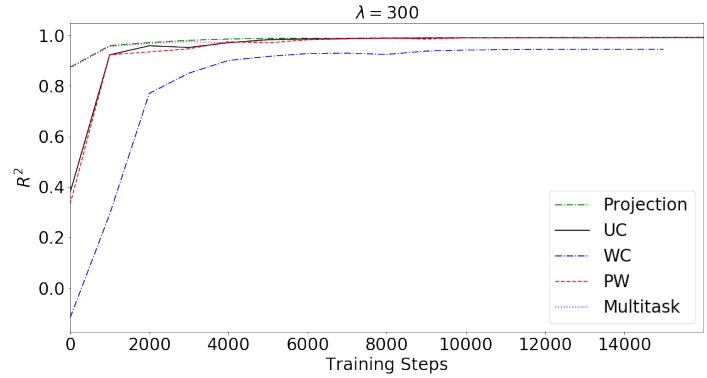
Figure 5.5: The trend of primary and auxiliary losses.

validation loss very early in the training process and after that they all start to over-fit after a while. However, as shown in Fig 5.4, the Pair-Wise based method is the most resilient one to over-fitting.

Fig 5.5a, Fig 5.5b, and Fig 5.5c illustrate how the primary loss and auxiliary losses are



(a) Average F-score for the sentiment analysis



(b) R^2 scores for the subjectivity analysis

Figure 5.6: The performance of each model on the auxiliary tasks.

evolving during the training. Surprisingly, the projection method provides the least decrease on the primary loss but it provides the most decrease on the auxiliary losses.

Overall, it seems that the Pair-Wise method has the most stable performance among all the other methods, by reaching to the lowest validation loss and achieving the lowest primary loss without significantly over-fitting to the model. It also has a good performance on the auxiliary tasks and in most of the cases has the same or a close performance to the Projection model. As shown in these figures, the multitask learning is not able to beat the performance of the Pair-Wise model and is lagging behind both in terms of the validation loss and auxiliary losses.

Extractive Summarization

Table 5.1 shows the performance of each model for the extractive summarization. In the single-task, we only train our model using the primary loss, while in multi-task learning with linearly combine the primary and auxiliary losses using λ. As shown in this table and

Table 5.1: The ROUGE score on the test dataset for CNN/DM dataset using $\lambda = 300$

	R-1	R-2	R-L
Single Task	40.56	18.09	37.07
Multi-Task	40.39	17.88	36.93
Projection	40.44	17.95	37.25
Unweighted Cosine	40.19	17.71	36.79
Weighted Cosine	40.45	17.93	37.04
Pair-Wise	**40.54**	**18.02**	**37.06**

following our insights in previous section, we observe that the Pair-Wise (PW) approach achieves the best performance among all the other methods, achieving a similar performance to our single-task summarization. However, as shown in this table, a large value of λ will definitely hurt the performance of the primary task. This is due to the fact that the overall loss now is far closer to the auxiliary task losses and its not putting much attention to the primary task. Although we have chosen a large value for λ and focusing highly on the auxiliary tasks, it is still interesting that the auxiliary training is able to achieve the same result as the single-task learning. Also, note that in this case, the multi-task learning has the worst performance comparing to other methods and its underperforming the single task model.

Auxiliary Tasks Performance

We also conduct another analysis to understand the performance of each auxiliary tasks during training. Fig 5.6a and Fig 5.6b shows the average F-score and R^2 score for the sentiment analysis and subjectivity analysis, respectively. As shown in these figures, almost all models perform well on these tasks, except the Weighted Cosine method. This directly comes from the fact that we are putting most of the attention of our training on these tasks. However, as we will observe in the following section, the performance of these models will drop significantly once we reduce the value of λ.

Effect of λ

The only hyper-parameter that decides the effect of auxiliary tasks on the primary task loss is the λ. For this experiment, we use two different values for λ. All experiments that are shown in previous section have a $\lambda = 300$, showing that we are highly relying on the auxiliary training in leading the primary task loss. In this section, we run another experiment with $\lambda = 1$. Note that when $\lambda = 1$ the primary loss and auxiliary losses will have the same importance in the overall loss. Moreover, the only difference between this case and the multitask learning is the fact that in auxiliary training, we modify the gradient of auxiliary tasks when they do not follow the direction of the primary task gradient, while

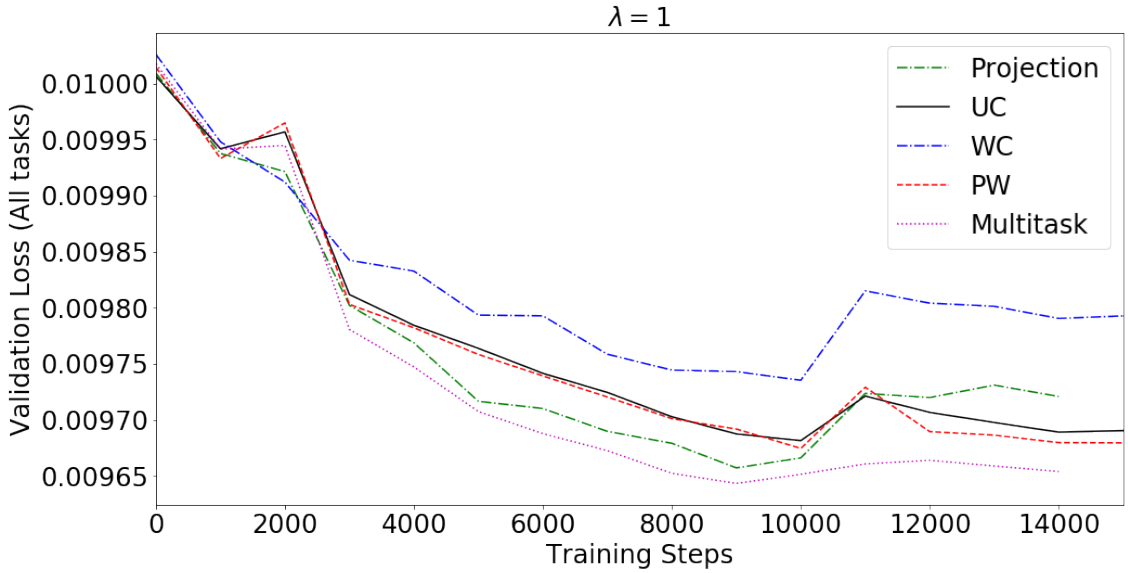


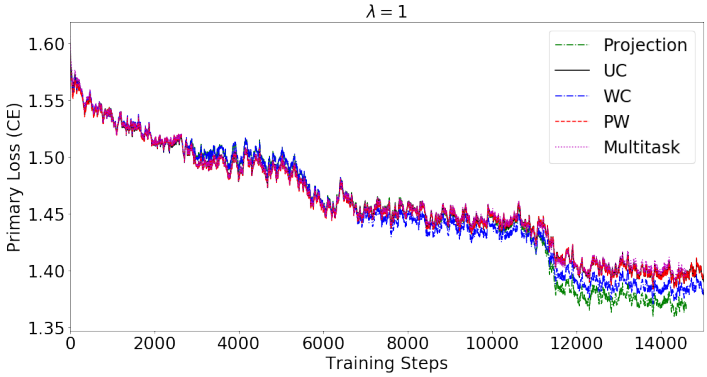
Figure 5.7: Validation Loss for the extractive text summarization during the auxiliary training with each method when $\lambda = 1$

in the multitask learning, we will not make these modifications on the auxiliary gradients.

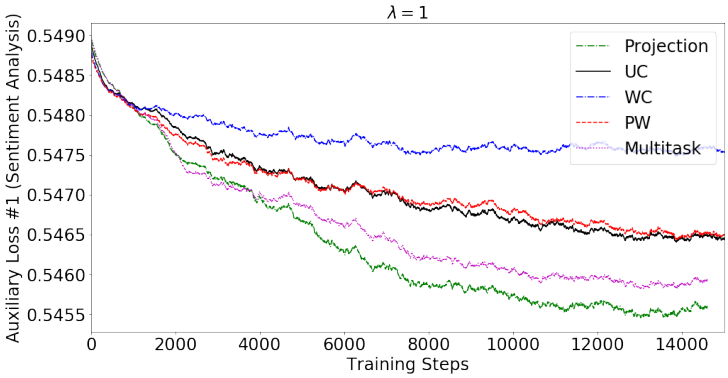
Similar to Fig 5.4, Fig 5.7 shows the validation loss when $\lambda = 1$. As shown in this figure and similar to the case where $\lambda = 300$, almost all models start over-fitting after a while with Weighted-Cosine having the most over-fitting. Moreover, similar to Fig 5.4 where the Pair-Wise model had the steadiest performance throughout the training, when $\lambda = 1$, its the multitask learning that provides the steadiest performance w.r.t. the validation loss. Although when we look at the Fig 5.8a it is the Projection model that provides the best loss for all tasks. This confirms that the multitask learning method over-fits on the training data. As shown in this figure, Projection based method is performing better than all the other methods and the multitask learning.

Fig 5.8b and Fig 5.8c show the trend of the auxiliary losses during training. As shown in these figures and unlike to the case where $\lambda = 300$, Projection method provides the best performance among other methods and as shown here, the Weighted Cosine (WC) provide the worst performance among others. Moreover, the difference between the WC and all other methods are more perceptible in these figures which shows how this method is vulnerable to the value of λ.

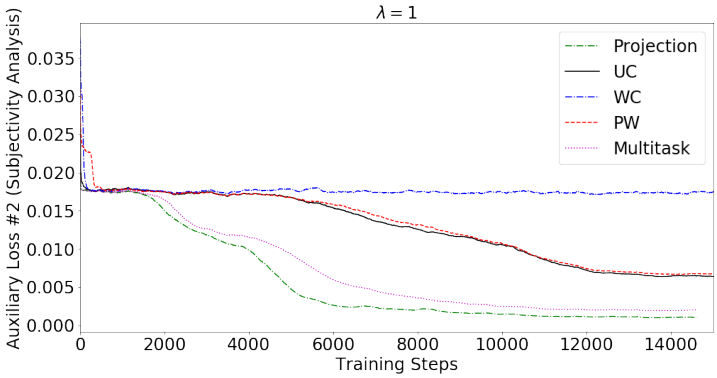
We will also evaluate the performance of each method on the extractive summarization on the CNN/DM dataset using each of these methods. Table 5.2 shows the performance of each model, when $\lambda = 1$. As shown in this table, all models perform better that their counterparts in Table 5.1. This is not surprising since by increasing the value of λ, the overall loss will be closer to the summation of auxiliary losses while by decreasing the λ



(a) Primary Loss when $\lambda = 1$



(b) Sentiment analysis loss when $\lambda = 1$



(c) Subjectivity analysis loss when $\lambda = 1$

Figure 5.8: The trend of primary and auxiliary losses when $\lambda = 1$.

value, the model will focus more on the primary loss. Although using a smaller value of λ seems to hurt the training of the whole model and specially auxiliary tasks, in general by putting more attention on the primary task, we can achieve a far better result. Comparing these results to the single-task training performance, we are improving it by %0.54 according

Table 5.2: The ROUGE score on the test dataset for CNN/DM dataset using $\lambda = 1$

	R-1	R-2	R-L
Single Task	40.56	18.09	37.07
Multi-Task	40.64	18.12	37.25
Projection	**41.10**	**18.58**	**37.66**
Unweighted Cosine	40.92	18.41	37.41
Weighted Cosine	40.90	18.38	37.46
Pair-Wise	40.97	18.44	37.50

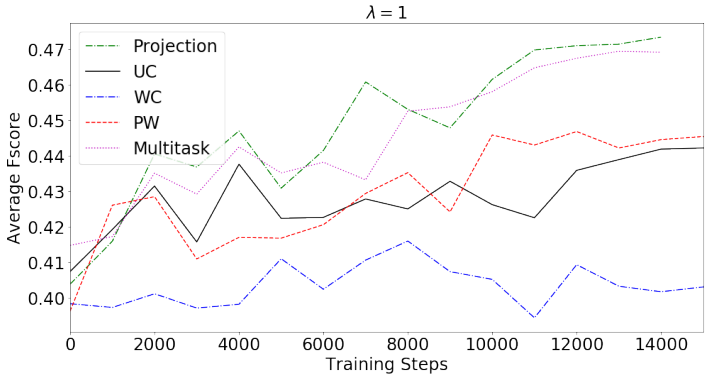
to the $R-1$ score which is statistically significant on this dataset. Moreover, as shown in this table, although the multi-task learning improves the result of the single-task learning, the improvement is not statistically significant and it is inferior to the projection method used for auxiliary training.

Finally, we evaluate the performance of the whole model on each auxiliary tasks. Fig 5.9a and Fig 5.9b shows this analysis. As shown in these figures and discussed earlier, when we lower the effect of λ for auxiliary task, the performance of these model drop and we see improvement on the primary task.

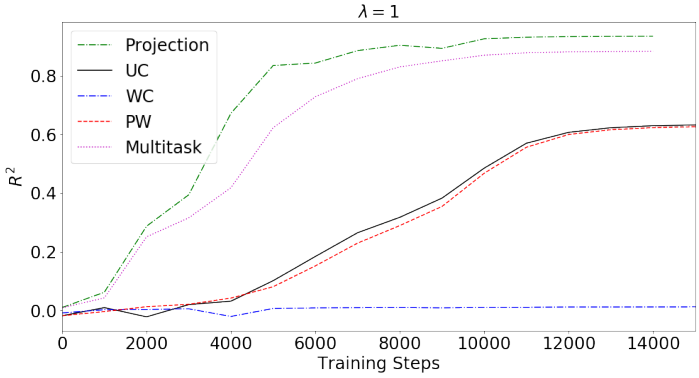
One of the most important insight from this table is that, using auxiliary training will improve the result of our primary task in all scenarios. This is important to note, since training the auxiliary task does not provide any overhead for the training and does not slow down the convergence of the model. Thus, it is evident that we should always benefit from external signals that could potentially help the performance of our primary task.

5.4 Conclusion

In this chapter, we studied using auxiliary training to improve the result of extractive summarization. Multi-task learning based models usually suffer from a phenomenon called negative transfer which adversely affect the training of the overall model and ultimately hurts the performance of each task. On the other hand, auxiliary training provides more freedoms for our models to avoid this phenomenon by ignoring the contribution of every tasks that impair the training of the a primary model. In this chapter, we focused on the extractive summarization problem and studied various solutions for the auxiliary training on this task. We showed how each method improve the performance of our primary task and compared the performance with the multi-task learning. According to our results, no matter what solution is used for auxiliary training, it consistently provides a far better result that the single task learning.



(a) Average F-score for the sentiment analysis



(b) R^2 scores for the subjectivity analysis

Figure 5.9: The performance of each model on the auxiliary tasks.

Chapter 6

Conclusion and Future Works

Modern news agencies are adapting to smart technologies faster than before. This comes as no surprise since the amount of content generated by these agencies is growing rapidly everyday and previous models that heavily rely on human interaction will no longer work for this massive amount of data. On the other hand, with this growth in the data, new opportunities will arise in understanding the type of content people are interested and improving them according to the interest of readers.

In this dissertation, we have presented various tools for handling and managing these massive information offered by news agencies and provided automated tools to facilitate some of the harder problems that traditionally could only be done by humans. Specifically, we have focused on two major problems as follows:

1. **Predicting popularity of news article**: In this problem, we were interested in understanding how each news article perform after its publication and how we can provide a framework that provide accurate prediction on the performance of the unpublished articles. To achieve this framework, we tackled this problem from different angles by first identifying the potential click-pattern of a times series and providing prediction as to when an article reaches its maximum number of viewers while also developed a prediction model that forecasts the long-term click-rate of a news article by only using information collected after 30 minutes of its publication.

2. **Automatic text summarization**: In this problem, we dealt with a harder problem which involves building a framework that understands human languages on long documents for the purpose of generating summary sentences for a news article. We have developed models based on deep neural networks which reads a news article and select the most salient sentences in the document and provide summary sentences accordingly for each article. However, due to the fact that these models require a huge number of training samples and have poor generalization to other news related dataset, we proposed various solutions for dealing with this problem. We first tackled the generalization problem by proposing a framework that tries to learn a generalized distribution using a training algorithm that uses two different text summarization datasets through a shared objective function.

In the past few years, text summarization using Deep Neural Networks has gained a lot of interests in the machine learning community. However, most of the successes in this area are

on extractive text summarization by selecting the most salient sentence from an article. On the other hand, human summaries have an abstractive nature and most of the summaries that exist in open-source text summarization datasets have this abstractive nature. Although abstractive summarization has been well-studied for text summarization problem, most of the best performing models (including our proposed model in this dissertation) are highly extractive. Therefore, the future works on this task will need to focus on models that could generate more novel words for an article. This could be achieved with better and more human-oriented paraphrasing model.

Moreover, current open-source datasets contain a lot of noisy and low quality summaries. Therefore, a framework that uses all these data for building a summarization model will consequently have trouble in convergence and producing higher quality results. Thus, putting more focus on selecting the right training samples efficiently is a necessity for current text summarization models.

Bibliography

[1] Shane Alcock and Richard Nelson. Application flow control in youtube video streams. *ACM SIGCOMM Computer Communication Review*, 41(2):24–30, 2011.

[2] Andreas Argyriou, Andreas Maurer, and Massimiliano Pontil. An algorithm for transfer learning in a heterogeneous environment. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 71–85. Springer, 2008.

[3] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *NIPS*, pages 2654–2662, 2014.

[4] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. *arXiv preprint arXiv:1412.7755*, 2014.

[5] Bart Bakker and Tom Heskes. Task clustering and gating for bayesian multitask learning. *Journal of Machine Learning Research*, 4(May):83–99, 2003.

[6] Roja Bandari, Sitaram Asur, and Bernardo A Huberman. The pulse of news in social media: Forecasting popularity. *ICWSM*, 12:26–33, 2012.

[7] Roja Bandari, Sitaram Asur, and Bernardo A Huberman. The pulse of news in social media: Forecasting popularity. In *ICWSM*, pages 26–33, 2012.

[8] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the ACL workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72, 2005.

[9] Shai Ben-David and Reba Schuller. Exploiting task relatedness for multiple task learning. In *Learning Theory and Kernel Machines*, pages 567–580. Springer, 2003.

[10] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *NIPS*, pages 1171–1179, 2015.

[11] Jonah Berger. Arousal increases social transmission of information. *Psychological science*, 22(7):891–893, 2011.

[12] Jonah Berger and Katherine L Milkman. What makes online content viral? *Journal of marketing research*, 49(2):192–205, 2012.

[13] Jonah Berger and Katy Milkman. Social transmission, emotion, and the virality of online content. *Wharton research, paper 106*, 2010.

[14] Jonah Berger and Eric M Schwartz. What drives immediate and ongoing word of mouth? *Journal of Marketing Research*, 48(5):869–880, 2011.

[15] Luca Bertinetto, João F Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. In *NIPS*, pages 523–531, 2016.

[16] Youmna Borghol, Sebastien Ardon, Niklas Carlsson, Derek Eager, and Anirban Mahanti. The untold story of the clones: content-agnostic factors that impact youtube video popularity. In *KDD*, pages 1186–1194. ACM, 2012.

[17] Ziqiang Cao, Wenjie Li, Sujian Li, Furu Wei, and Yanran Li. Attsum: Joint learning of focusing and summarization with neural attention. In *COLING*, pages 547–556, 2016.

[18] Rich Caruana, Shumeet Baluja, and Tom Mitchell. Using the future to" sort out" the present: Rankprop and multitask learning for medical risk evaluation. In *Advances in neural information processing systems*, pages 959–965, 1996.

[19] Asli Celikyilmaz, Antoine Bosselut, Xiaodong He, and Yejin Choi. Deep communicating agents for abstractive summarization. In *NAACL-HLT*, volume 1, pages 1662–1675, 2018.

[20] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. Analyzing the video popularity characteristics of large-scale user generated content systems. *Ieee/Acm Transactions On Networking (Ton)*, 17(5):1357–1370, 2009.

[21] Yen-Chun Chen and Mohit Bansal. Fast abstractive summarization with reinforce-selected sentence rewriting. In *ACL*, volume 1, pages 675–686, 2018.

[22] Jianpeng Cheng and Mirella Lapata. Neural summarization by extracting sentences and words. In *ACL*, volume 1, pages 484–494, 2016.

[23] Justin Cheng, Lada Adamic, P Alex Dow, Jon Michael Kleinberg, and Jure Leskovec. Can cascades be predicted? In *WWW*, pages 925–936. ACM, 2014.

[24] Sumit Chopra, Michael Auli, and Alexander M Rush. Abstractive sentence summarization with attentive recurrent neural networks. In *NAACL-HLT*, pages 93–98, 2016.

[25] William S Cleveland and Susan J Devlin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of the American statistical association*, 83 (403):596–610, 1988.

[26] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, pages 647–655, 2014.

[27] Bonnie Dorr, David Zajic, and Richard Schwartz. Hedge trimmer: A parse-and-trim approach to headline generation. In *Proceedings of the HLT-NAACL on Text summarization workshop-Volume 5*, pages 1–8. ACL, 2003.

[28] Yunshu Du, Wojciech M Czarnecki, Siddhant M Jayakumar, Razvan Pascanu, and Balaji Lakshminarayanan. Adapting auxiliary losses using gradient similarity. *arXiv preprint arXiv:1812.02224*, 2018.

[29] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *NIPS*, pages 1087–1098, 2017.

[30] William H DuBay. The principles of readability. *Online Submission*, 2004.

[31] Eric Eaton, Terran Lane, et al. Modeling transfer relationships between learning tasks for improved inductive transfer. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 317–332. Springer, 2008.

[32] Angela Fan, David Grangier, and Michael Auli. Controllable abstractive summarization. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 45–54. ACL, 2018.

[33] Zhe Gan, Yunchen Pu, Ricardo Henao, Chunyuan Li, Xiaodong He, and Lawrence Carin. Learning generic sentence representations using convolutional neural networks. In *EMNLP*, pages 2390–2400, 2017.

[34] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

[35] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *ICML*, pages 1243–1252, 2017.

[36] Sebastian Gehrmann, Yuntian Deng, and Alexander M Rush. Bottom-up abstractive summarization. *arXiv preprint arXiv:1808.10792*, 2018.

[37] Max Grusky, Mor Naaman, and Yoav Artzi. Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies. In *NAACL-HLT*, 2018.

[38] Gonca Gürsun, Mark Crovella, and Ibrahim Matta. Describing and forecasting video access patterns. In *IEEE INFOCOM*, pages 16–20. IEEE, 2011.

[39] Di He, Hanqing Lu, Yingce Xia, Tao Qin, Liwei Wang, and Tieyan Liu. Decoding with value networks for neural machine translation. In *NIPS*, pages 177–186, 2017.

[40] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *NIPS*, pages 1693–1701, 2015.

[41] Wan-Ting Hsu, Chieh-Kai Lin, Ming-Ying Lee, Kerui Min, Jing Tang, and Min Sun. A unified model for extractive and abstractive summarization using inconsistency loss. In *ACL*, pages 132–141, 2018.

[42] Minghao Hu, Yuxing Peng, and Xipeng Qiu. Reinforced mnemonic reader for machine comprehension. *CoRR, abs/1705.02798*, 2017.

[43] CJ Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *AAAI*, 2014.

[44] Clayton J Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *AAAI*, 2014.

[45] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*, 2017.

[46] Yaser Keneshloo, Shuguang Wang, Eui-Hong (Sam) Han, and Naren Ramakrishnan. Predicting the popularity of news articles. In *SIAM SDM*, 2016.

[47] Yaser Keneshloo, Tian Shi, Naren Ramakrishnan, and Chandan K. Reddy. Deep reinforcement learning for sequence to sequence models. *arXiv:1805.09461*, 2018.

[48] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, pages 1746–1751, 2014.

[49] Jon Kleinberg. Bursty and hierarchical structure in streams. *Data Mining and Knowledge Discovery*, 7(4):373–397, 2003.

[50] Y. F. Kong, Shoubin and L. Feng. Predicting future retweet counts in a microblog. *Journal of Computational Information Systems*, 10(4):1393–1404, 2014.

[51] Julia Kreutzer, Joshua Uyheng, and Stefan Riezler. Reliability and learnability of human bandit feedback for sequence-to-sequence reinforcement learning. In *ACL*, volume 1, pages 1777–1788, 2018.

[52] Wojciech Kryściński, Romain Paulus, Caiming Xiong, and Richard Socher. Improving abstraction in text summarization. *arXiv preprint arXiv:1808.07913*, 2018.

[53] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *WWW*, pages 591–600. AcM, 2010.

[54] Jong Gun Lee, Sue Moon, and Kavé Salamatian. Modeling and predicting the popularity of online contents with cox proportional hazard regression model. *Neurocomputing*, 76(1):134–145, 2012.

[55] Kristina Lerman and Rumi Ghosh. Information contagion: An empirical study of the spread of news on digg and twitter social networks. *ICWSM*, 10:90–97, 2010.

[56] Kristina Lerman and Tad Hogg. Using a model of social dynamics to predict popularity of news. In *WWW*, pages 621–630. ACM, 2010.

[57] Chen Li, Fei Liu, Fuliang Weng, and Yang Liu. Document summarization via guided sentence compression. In *EMNLP*, pages 490–500, 2013.

[58] Haoran Li, Junnan Zhu, Jiajun Zhang, and Chengqing Zong. Ensure the correctness of the summary: incorporate entailment knowledge into abstractive sentence summarization. In *ICML*, pages 1430–1441, 2018.

[59] Jiwei Li, Will Monroe, and Dan Jurafsky. Learning to decode for future success. *arXiv preprint arXiv:1701.06549*, 2017.

[60] Piji Li, Wai Lam, Lidong Bing, and Zihao Wang. Deep recurrent generative decoder for abstractive text summarization. In *EMNLP*, pages 2091–2100, 2017.

[61] Piji Li, Lidong Bing, and Wai Lam. Actor-critic based training framework for abstractive summarization. *arXiv preprint arXiv:1803.11070*, 2018.

[62] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[63] Zichao Li, Xin Jiang, Lifeng Shang, and Hang Li. Paraphrase generation with deep reinforcement learning. *arXiv preprint arXiv:1711.00279*, 2017.

[64] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *ACL*, volume 1, pages 23–33, 2017.

[65] Bill Yuchen Lin and Wei Lu. Neural adaptation layers for cross-domain named entity recognition. In *EMNLP*, pages 2012–2022, 2018.

[66] C-Y LIN. Rouge: A package for automatic evaluation of summaries. In *Proc. of Workshop on Text Summarization Branches Out*, 2004.

[67] Cédric Lopez, Violaine Prince, and Mathieu Roche. How can catchy titles be generated without loss of informativeness? *Expert Systems with Applications*, 41(4):1051–1062, 2014.

[68] MM Mahmud and Sylvian Ray. Transfer learning using kolmogorov complexity: Basic theory and empirical evaluations. In *NIPS*, pages 985–992, 2008.

[69] Luís Marujo, Miguel Bugalho, João Paulo da Silva Neto, Anatole Gershman, and Jaime Carbonell. Hourly traffic prediction of news stories. *arXiv preprint arXiv:1306.4608*, 2013.

[70] Yasuko Matsubara, Yasushi Sakurai, B Aditya Prakash, Lei Li, and Christos Faloutsos. Rise and fall patterns of information diffusion: model and implications. In *KDD*, pages 6–14. ACM, 2012.

[71] Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The natural language decathlon: Multitask learning as question answering. *arXiv:1806.08730*, 2018.

[72] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[73] Terence C. Mills. *Time series techniques for economists*. Cambridge University Press, Cambridge, 1990.

[74] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *ICLR*, 2016.

[75] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[76] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529, 2015.

[77] Taylor Mordan, Nicolas Thome, Gilles Henaff, and Matthieu Cord. Revisiting multitask learning with rock: a deep residual auxiliary block for visual detection. In *Advances in Neural Information Processing Systems*, pages 1310–1322, 2018.

[78] Seth A Myers and Jure Leskovec. The bursty dynamics of the twitter information network. In *WWW*, pages 913–924. ACM, 2014.

[79] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence rnns and beyond. In *SIGNLL*, pages 280–290, 2016.

[80] Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In *AAAI*, pages 3075–3081, 2017.

[81] Shashi Narayan, Nikos Papasarantopoulos, Shay B Cohen, and Mirella Lapata. Neural extractive summarization with side information. *arXiv preprint arXiv:1704.04530*, 2017.

[82] Shashi Narayan, Shay B Cohen, and Mirella Lapata. Ranking sentences for extractive summarization with reinforcement learning. In *NAACL-HLT*, 2018.

[83] Jun-Ping Ng, Praveen Bysani, Ziheng Lin, Min-Yen Kan, and Chew-Lim Tan. Exploiting category-specific information for multi-document summarization. *COLING*, pages 2093–2108, 2012.

[84] Boyuan Pan, Yazheng Yang, Zhou Zhao, Yueting Zhuang, Deng Cai, and Xiaofei He. Discourse marker augmented network with reinforcement learning for natural language inference. In *ACL*, volume 1, pages 989–999, 2018.

[85] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE TKDE*, 22 (10):1345–1359, 2009.

[86] John Paparrizos and Luis Gravano. k-shape: Efficient and accurate clustering of time series. In *ICDM*, pages 1855–1870. ACM, 2015.

[87] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *ACL*, pages 311–318, 2002.

[88] Nish Parikh and Neel Sundaresan. Scalable and near real-time burst detection from ecommerce queries. In *KDD*, pages 972–980. ACM, 2008.

[89] Ramakanth Pasunuru and Mohit Bansal. Multi-reward reinforced summarization with saliency and entailment. In *NAACL-HLT*, volume 2, pages 646–653, 2018.

[90] Romain Paulus, Caiming Xiong, and Richard Socher. A deep reinforced model for abstractive summarization. In *ICLR*, 2018.

[91] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[92] Henrique Pinto, Jussara M Almeida, and Marcos A Gonçalves. Using early view patterns to predict the popularity of youtube videos. In *WSDM*, pages 365–374. ACM, 2013.

[93] Edoardo Maria Ponti, Ivan Vulić, Goran Glavaš, Nikola Mrkšić, and Anna Korhonen. Adversarial propagation and zero-shot cross-lingual transfer of word vector specialization. In *EMNLP*, pages 282–293. ACL, 2018. URL http://aclweb.org/anthology/D18-1026.

[94] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

[95] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2016.

[96] Julio Reis, Fabricio Benevenuto, Raquel Prates, Haewoon Kwak, and Jisun An. Breaking the news: First impressions matter on online news. In *AAAI*, 2015.

[97] Pengjie Ren, Zhumin Chen, Zhaochun Ren, Furu Wei, Liqiang Nie, Jun Ma, and Maarten De Rijke. Sentence relations for extractive summarization with deep neural networks. *ACM TOIS*, 36(4):39, 2018.

[98] Steven J Rennie, Etienne Marcheret, Youssef Mroueh, Jerret Ross, and Vaibhava Goel. Self-critical sequence training for image captioning. In *CVPR*, pages 7008–7024, 2017.

[99] Michael T Rosenstein. To transfer or not to transfer. In *In workshop on transfer learning*, volume 898, pages 1–4, 2005.

[100] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[101] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *EMNLP*, pages 379–389, 2015.

[102] Devendra Singh Sachan, Pengtao Xie, and Eric P Xing. Effective use of bidirectional language modeling for medical named entity recognition. *arXiv, arXiv:1711.07908*, 2017.

[103] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *ICML*, pages 1842–1850, 2016.

[104] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

[105] Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. In *ACL*, volume 1, pages 1073–1083, 2017.

[106] Tushar Semwal, Promod Yenigalla, Gaurav Mathur, and Shivashankar B Nair. A practitioners' guide to transfer learning for text classification using convolutional neural networks. In *SDM*, pages 513–521. SIAM, 2018.

[107] Tian Shi, Yaser Keneshloo, Naren Ramakrishnan, and Chandan K Reddy. Neural abstractive text summarization with sequence-to-sequence models. *arXiv preprint arXiv:1812.02303*, 2018.

[108] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NIPS*, pages 4077–4087, 2017.

[109] Shashidhar Sundereisan, Abhay Bhadriraju, M Saquib Khan, Naren Ramakrishnan, and B Aditya Prakash. Sanstext: Classifying temporal topic dynamics of twitter cascades without tweet text. In *ASONAM*, pages 649–656. IEEE, 2014.

[110] Gabor Szabo and Bernardo A Huberman. Predicting the popularity of online content. *Communications of the ACM*, 53(8):80–88, 2010.

[111] Jiwei Tan, Xiaojun Wan, and Jianguo Xiao. Abstractive document summarization with a graph-based attentional neural model. In *ACL*, volume 1, pages 1171–1181, 2017.

[112] Alexandru Tatar, Panayotis Antoniadis, Marcelo Dias De Amorim, and Serge Fdida. Ranking news articles based on popularity prediction. In *ASONAM*, pages 106–110. IEEE, 2012.

[113] Alexandru Tatar, Marcelo Dias De Amorim, Serge Fdida, and Panayotis Antoniadis. A survey on predicting the popularity of web content. *Journal of Internet Services and Applications*, 5(1):8, 2014.

[114] Trieu Trinh, Andrew Dai, Thang Luong, and Quoc Le. Learning longer-term dependencies in rnns with auxiliary losses. In *International Conference on Machine Learning*, pages 4972–4981, 2018.

[115] Manos Tsagkias, Wouter Weerkamp, and Maarten De Rijke. Predicting the volume of comments on online news stories. In *CIKM*, pages 1765–1768. ACM, 2009.

[116] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *CVPR*, volume 1, page 4, 2017.

[117] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2016.

[118] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, et al. Tensor2tensor for neural machine translation. *arXiv:1803.07416*, 2018.

[119] Alexei Vázquez, Joao Gama Oliveira, Zoltán Dezsö, Kwang-Il Goh, Imre Kondor, and Albert-László Barabási. Modeling bursts and heavy tails in human dynamics. *Physical Review E*, 73(3):036127, 2006.

[120] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[121] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *NIPS*, pages 2692–2700, 2015.

[122] Li Wang, Junlin Yao, Yunzhe Tao, Li Zhong, Wei Liu, and Qiang Du. A reinforced topic-aware convolutional sequence-to-sequence model for abstractive text summarization. *arXiv preprint arXiv:1805.03616*, 2018.

[123] Lu Wang, Hema Raghavan, Vittorio Castelli, Radu Florian, and Claire Cardie. A sentence compression based framework to query-focused multi-document summarization. In *ACL*, volume 1, pages 1384–1394, 2013.

[124] Senzhang Wang, Zhao Yan, Xia Hu, S Yu Philip, Zhoujun Li, and Biao Wang. Cpb: a classification-based approach for burst time prediction in cascades. *Knowledge and Information Systems*, pages 1–29, 2015.

[125] Xuanhui Wang, ChengXiang Zhai, Xiao Hu, and Richard Sproat. Mining correlated bursty topic patterns from coordinated text streams. In *KDD*, pages 784–793. ACM, 2007.

[126] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.

[127] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.

[128] Y. Wu and B. Hu. Learning to Extract Coherent Summary via Deep Reinforcement Learning. *arXiv preprint arXiv:1804.07036*, 2018.

[129] Jaewon Yang and Jure Leskovec. Patterns of temporal variation in online media. In *WSDM*, pages 177–186. ACM, 2011.

[130] Michihiro Yasunaga, Rui Zhang, Kshitijh Meelu, Ayush Pareek, Krishnan Srinivasan, and Dragomir Radev. Graph-based neural multi-document summarization. In *CoNLL*, pages 452–462, 2017.

[131] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *NIPS*, pages 3320–3328, 2014.

[132] Xingdi Yuan, Tong Wang, Caglar Gulcehre, Alessandro Sordoni, Philip Bachman, Saizheng Zhang, Sandeep Subramanian, and Adam Trischler. Machine comprehension by text-to-text neural question generation. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 15–25, 2017.

[133] Tauhid Zaman, Emily B Fox, and Eric T Bradlow. A bayesian approach for predicting the popularity of tweets. *The Annals of Applied Statistics*, 8(3):1583–1611, 2014.

[134] Xingxing Zhang, Mirella Lapata, Furu Wei, and Ming Zhou. Neural latent extractive document summarization. In *EMNLP*, pages 779–784, 2018.

[135] Yuting Zhang, Kibok Lee, and Honglak Lee. Augmenting supervised neural networks with unsupervised objectives for large-scale image classification. In *ICLR*, pages 612–621, 2016.

[136] Zhanpeng Zhang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Facial landmark detection by deep multi-task learning. In *European conference on computer vision*, pages 94–108. Springer, 2014.

[137] Qingyuan Zhao, Murat A Erdogdu, Hera Y He, Anand Rajaraman, and Jure Leskovec. Seismic: A self-exciting point process model for predicting tweet popularity. In *KDD*, pages 1513–1522. ACM, 2015.

[138] Qingyu Zhou, Nan Yang, Furu Wei, and Ming Zhou. Selective encoding for abstractive sentence summarization. In *ACL*, volume 1, pages 1095–1104, 2017.

[139] Qingyu Zhou, Nan Yang, Furu Wei, Shaohan Huang, Ming Zhou, and Tiejun Zhao. Neural document summarization by jointly learning to score and select sentences. In *ACL*, pages 654–663. ACL, 2018.

[140] Yunyue Zhu and Dennis Shasha. Efficient elastic burst detection in data streams. In *KDD*, pages 336–345. ACM, 2003.