TÉCNICO LISBOA

Hybrid Extractive/Abstractive Summarization Using Pre-Trained Sequence-to-Sequence Models

David Ferreira Coimbra

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Bruno Emanuel Da Graça Martins
Prof. Rui Filipe Lima Maranhão de Abreu

Examination Committee

Chairperson: Prof. Mário Jorge Costa Gaspar da Silva
Supervisor: Prof. Bruno Emanuel Da Graça Martins
Member of the Committee: Prof. David Manuel Martins de Matos

November 2020

Acknowledgments

A sincere thank you to my supervisors, Prof. Bruno Martins and Prof. Rui Maranhão, for the regular support and guidance on a challenging topic that was completely new to me.

Abstract

Typical document summarization methods can be either extractive, by selecting appropriate parts of the input text to include in the summary, or abstractive, by generating new text with basis on a meaningful representation of the source text. In both cases, the current state-of-the-art involves the use of pre-trained neural language models based on the Transformer architecture. Most of these approaches are unable to process input text beyond a limited small number of tokens. This paper advances a hybrid summarization approach based on a single T5 model, which first selects important sentences from the source text, and subsequently produces an abstractive summary from the selected sentences. In doing this, we reduce the overall computational requirements associated to the use of Transformer models, and mitigate the effects of their input size limitations, while ensuring a good performance. Through experiments with different datasets, we show that our method achieves comparable results to current state-of-the-art models, while maintaining relatively low computational requirements.

Keywords: Single Document Summarization, Natural Language Generation, Sequence-to-Sequence Neural Models, Transfer Learning

Resumo

Os métodos típicos para resumo de documentos podem ser extrativos, que selecionam partes apropriadas do texto de entrada para incluir no resumo, ou abstrativos, que geram um novo texto com base numa representação do texto de origem. Em ambos os casos, o estado da arte atual envolve o uso de modelos neuronais de linguagem pré-treinados com base na arquitetura Transformer. A maioria destas abordagens não consegue processar o texto de entrada para além de um pequeno número limitado de tokens. Este estudo propõe uma abordagem de sumarização híbrida com base num único modelo T5, que primeiro seleciona frases importantes do texto de origem e, posteriormente, produz um resumo abstrativo a partir frases selecionadas. Assim, reduzimos os requisitos computacionais associados ao uso de modelos Transformer e mitigamos os efeitos das suas limitações de tamanho de sequências de entrada, garantindo um bom desempenho. Experimentando com conjuntos de dados diferentes, mostramos que nosso método atinge resultados comparáveis aos modelos atuais, mantendo requisitos computacionais relativamente baixos.

Palavras-chave: sumarização de documentos únicos, geração de linguagem natural, modelos neuronais sequência-a-sequência, *Transfer Learning*

Contents

List of Tables

List of Figures

Acronyms

Adam Adaptive moment estimation.

BERT Bidirectional Encoder Representations from Transformers.

CBOW Continuous Bag of Words.

ELMo Embeddings from Language Models.

GPU Graphics Processing Unit.

GRU Gated Recurrent Unit.

IDF Inverse Document Frequency.

LSTM Long Short Term Memory.

MLM Masked Language Modeling.

MLP Multi Layer Perceptron.

NLP Natural Language Processing.

NSP Next Sentence Prediction.

PEGASUS Pre-training with Extracted Gap-sentences for Abstractive Summarization.

RNN Recurrent Neural Network.

ROUGE Recall-Oriented Understudy for Gisting Evaluation.

SGD Stochastic Gradient Descent.

T5 Text-to-Text Transfer Transformer.

Chapter 1

Introduction

The internet has made it easy to share large amounts of information in a considerable variety of topics, formats, and styles. Notably, some of the most popular means of information sharing is the online forum, also known as the discussion board. These are designed for users to easily ask questions or problems, and allow other users to comment on and discuss any relevant topic or solution. For example, TripAdvisor is a website and mobile application that allows users to review hotels, restaurants and tourist destinations. Users are able to ask and discuss any question related to a particular destination or point of interest. Another example is found within GitHub, a popular version control platform for software projects catering to many purposes, including functionalities for requirements communication and bug reporting through its issue tracking system, which is implemented as a discussion board [Bissyandé et al., 2013]. This is a convenient and simple means to share information, and helpful to users who are faced with the same problem in the future, simply by finding the appropriate discussion and consulting comments from other users. However it becomes difficult to find the most important information among all posts in a conversation that may contain multiple points of view [Bhatia et al., 2014]. In these cases, a user may prefer reading a summary of the discussions, where less important information has been discarded to save time and reading effort.

In Natural Language Processing (NLP), the goal of automatic summarization is to produce a document of shorter length than the source document(s); preserving its overall meaning while discarding unimportant information. Text summarization methods focus on two approaches: extractive summarization [Nallapati et al., 2017] selects important fragments from the source document to be included in the final summary; while abstractive summarization [Rush et al., 2015, See et al., 2017] generates new, paraphrased text that succinctly represents the knowledge in the source document, an approach similar to the one present in humans. Hybrid methods [Liu et al., 2018, Liu and Lapata, 2019], which contain both extractive and abstractive modules, combine the advantages of both approaches.

Each of these strategies has its own advantages and limitations. In this work, we investigate the applicability of each approach to the task of summarizing online forum discussions, as well as its extensibility to more variable domains.

1.1 Thesis Statement

The motivation behind our research proposal is to compare different approaches to automatic summarization applied to the domain of online forum discussions, based on implementations of the Transformer architecture [Vaswani et al., 2017] and subsequent enhancements to different text classification and generation tasks. These models routinely achieve state-of-the-art performance in many summarization tasks, most notably on the popular CNN/DailyMail dataset for the news domain [Nallapati et al., 2016a], which consists of news articles paired with summaries written by the authors themselves. Few studies, however, have yet examined the task of summarizing online discussions [Tarnpradab et al., 2017].

The emergence of deep neural networks has led to numerous breakthroughs in various NLP tasks, powered by end-to-end deep neural models [Kryscinski et al., 2019]. In particular, pre-trained Transformer-based models [Devlin et al., 2019, Lewis et al., 2020, Raffel et al., 2020] registered state-of-the-art results in sequence classification, question answering, named entity recognition and language translation, as well as extractive and abstractive summarization. However, training of Transformer models has notoriously large memory requirements due to increasing parameter count, becoming limited by the available memory in the device(s) used for training [Sohoni et al., 2019]. As the dimensionality and effective size of datasets continues to increase, this forces researchers to use less expressive models or to downsample training inputs to a lower dimensionality.

Recent studies [Liu et al., 2018, Subramanian et al., 2019, Tretyak and Stepanov, 2020] have attempted to enhance state-of-the-art summarization models by combining methods from extractive and abstractive approaches, conditioning an abstractive model with an extracted summary of the source document. This serves as an additional source of information for the abstractive model, allowing it to differentiate information relevancy better. Not only has this approach been shown to lead to better results than extractive or abstractive summarization alone, it also has the consequence of reducing the size of each sample in the source dataset. As such, it provides a way of selecting which information to exclude when reducing the length of an input sample, more effective than simply truncating the document to its beginning. Moreover, Transformer-based models perform well in a variety of NLP tasks due to benefiting from being pre-trained on large English corpora. As such, it is possible to train and use the same model on extractive and abstractive summarization tasks simultaneously.

Following this line of work, we experiment with hybrid methods for iteratively training a summarization model in both extractive and abstractive summarization tasks in the domain of online discussions. We hypothesize that iteratively conditioning each training procedure with information from the previous one, the model will learn to select and exclude important information more effectively, minimizing the performance loss associated with memory management. Thus, this study aims at answering the following research question: RQ-I) *Which approach leads to better results in an online discussions domain?*, i.e. if our iterative hybrid method results in higher metric scores than standalone extractive or abstractive methods). RQ-II) *Is this approach applicable to other domains?*, i.e. verify if the benefit holds for datasets with more samples, longer texts and higher grammatical variety.

1.2 Methodology

The main objective of this proposal is to create a unified, iterative training procedure that leverages pre-trained sequence-to-sequence models to combine extractive and abstractive approaches to text summarization, applied to the domain of summarizing online forum discussions. To this aim, we conducted the majority of our study on the TripAdvisor dataset [Bhatia et al., 2014, Tarnpradab et al., 2017], as it is the most popular corpus among studies centered on the online discussions domain, consisting of a collection of user-generated discussion threads centered around tourist destinations in New York City.

The challenges faced in this work, in addition to those found in natural language processing, were to find an appropriate framework for conducting experiments with various existing models. Before conducting any experiments, it was necessary to study and compare different state-of-the-art models, documenting their effectiveness and applicability to summarization tasks. This is crucial to establish a baseline to which results can be compared, so we can assess which enhancements to the training procedure lead to better results, thus be able to answer RQ-I.

In order to answer RQ-II, special effort was given to finding a means to train on large datasets. It was necessary to lazily load batches of sentences from a very large file on disk while maintaining the ability to randomize the samples in each batch. This is to maintain a level of consistency across experiments with different datasets. The dataset chosen for this task was the arXiv dataset [Cohan et al., 2018], as it is popularly used for tasks centered on very long documents. For this dataset of scientific articles, the summarization task aims to predict the abstract section based on the document body.

The main Transformer-based model that was considered for our experiments was the Text-to-Text Transfer Transformer (T5) [Raffel et al., 2020]. We chose this model for its unified text-to-text interface that allows us to train it in several tasks iteratively, without a need to change layers or hyperparameters. We leveraged this characteristic to develop a fine-tuning routine to iteratively train a single model in both extractive and abstractive summarization tasks.

To conduct our experiments, we took advantage of the popular open-source library Transformers by Hugging Face [Wolf et al., 2019]. Transformers is a Python library which provides an easy to use API for interacting with various state-of-the-art Transformer models. It also provides a convenient toolkit for pre-processing and tokenizing textual data. We chose to use this library with PyTorch [Paszke et al., 2019] as a framework for our experiments, as it provides abstractions that make the baseline training procedure more easily reproducible and extensible. Furthermore, to reduce a large amount of boilerplate code related to loading and batching data and performing operations on the studied models, which would be unnecessarily repeated in all experiments, we leveraged the PyTorch Lightning[1] framework. Pandas [McKinney, 2010] was used to programatically load and access data samples.

However, at every step along the way, the biggest overarching problem were computational resources, especially memory limitations and time constraints, which placed a hard limit on the usable methods and their parameters. With larger datasets, it becomes too time-consuming to train Transformer models for long enough to produce results comparable to state-of-the-art. Consequently, and

[1] `https://pytorch-lightning.readthedocs.io`

due to the cloud-based nature of many deep learning platforms, and the unsuitability of consumer hardware, many processes had to be severely altered to fit this context. After conducting a few experiments on Google Colaboratory[2], we settled on a remote virtual machine provided by Google Cloud Platform's Compute Engine[3], equipped with an Nvidia Tesla T4 GPU.

1.3 Results and Contributions

The main contribution of this work is an experimental comparative study that explores the applicability of different fine-tuning approaches in the domain of summarization of online discussions, powered by a pre-trained Transformer-based sequence-to-sequence model. Moreover, we present a novel fine-tuning approach where the model iteratively learns to extract important sentences from a document and subsequently generate an abstractive summary from the extracted sentences. The final model can be used for both extractive and abstractive summarization. Overall performance is evaluated through standard evaluation metrics for text summarization: a ROUGE score [Lin, 2004] for comparison with state-of the art, and BERTScore [Zhang et al., 2020] for a finer assessment of content adequacy.

1.4 Document Overview

This document is organized as follows:

- Chapter 2 describes fundamental theoretical concepts and related work that lay the foundation for the methods and approaches considered in this study. We start with an overview of the fundamental neural models for machine learning in section 2.1, as well as the methods used to represent textual information for these models. Section 2.4 provides a bibliographic review of summarization approaches powered by neural models, concluding with an overview of major studies focusing on the summarization of online discussions.
- Chapter 3 describes our experiments, as well as our hybrid iterative training procedure for summarization.
- Chapter 4 presents the experimental evaluation of each experience as well as the effectiveness of our novel method. Section 4.1 describes a statistical analysis of the datasets used in the experiments, together with evaluation metrics. Section 4.2 presents the final results, with corresponding numerical scores.
- Finally, chapter 5 describes the main contributions of this study and outlines possible enhancements to be explored in future work.

[2] https://research.google.com/colaboratory/faq.html
[3] https://cloud.google.com/compute/docs

Chapter 2

Fundamental Concepts and Related Work

This chapter focuses on general concepts pertaining to machine learning and natural language processing, which make up the foundations of the proposed research project, as well how these concepts have been applied in the domain of text summarization, and online discussions in particular. Section 2.1 describes fundamental structures in deep learning: the perceptron, the multi layer perceptron, and recurrent neural networks, as well as the methods used for training these structures. Section 2.2 goes into detail on approaches for text representation, starting with one-hot vectors and moving onto dense representations, both context-free and contextual. This section ends with an introduction to deep bidirectional models taking advantage of the Transformer architecture. Section 2.4 presents a detailed review of recent works pertaining to the task of text summarization using deep learning techniques for natural language processing, as well as relevant applications of NLP to tasks related to the understanding and summarization of online discussions.

2.1 Machine Learning with Neural Networks

An artificial neural network is a computing system that is inspired by biological brains [Goldberg, 2017]. It is composed of a set of computation units, called neurons, that are connected to each other in a network. Neurons are arranged in layers, and a neural network can be seen as having an input layer, an output layer, and one or more *hidden* layers. The values of each layer can be represented by a vector. The input layer is a d_{in}-dimensional vector $\boldsymbol{x}$, and each hidden layer $\boldsymbol{h}$ is the result of a linear transformation $\boldsymbol{W}\boldsymbol{x}$, where $\boldsymbol{W}$ is a $d_{in} \times d_h$ matrix specifying the weights of a given layer.

2.1.1 Perceptrons

The perceptron [Rosenblatt, 1958] is the simplest implementation of a neural network, corresponding to a single computation unit. It takes as input a vector $\boldsymbol{x}$ (the input layer) and produces an output vector

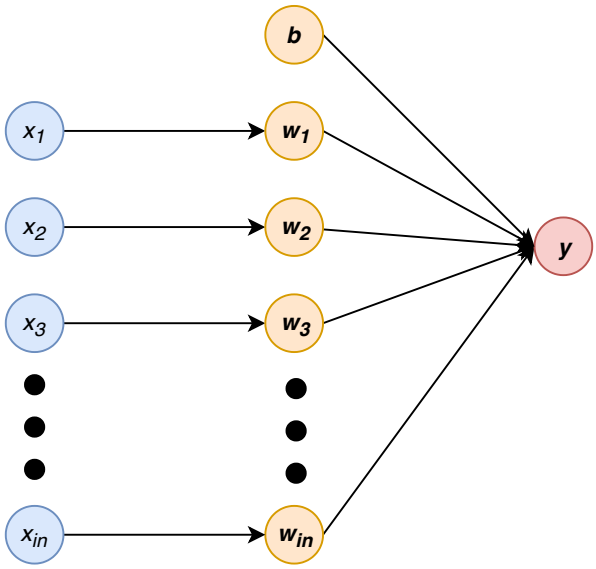


Figure 2.1: An illustration of the perceptron model, where each blue neuron is a scalar element of the input vector $\boldsymbol{x}$. Each yellow neuron is a row vector in the weight matrix $\boldsymbol{W}$, with an additional bias $\boldsymbol{b}$. The red neuron is the output vector $\boldsymbol{y}$.

$\boldsymbol{y}$ (the output layer). Each input is associated with a weight vector, which is a row in a matrix $\boldsymbol{W}$. The perceptron, illustrated in Figure 2.1, is thus a simple linear model, corresponding to

$$\mathrm{NN}_{\mathrm{Perceptron}}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{2.1}$$

where $\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ is the input, $\boldsymbol{W} \in \mathbb{R}^{d_{in} \times d_{out}}$ is the weight matrix and $\boldsymbol{b} \in \mathbb{R}^{d_{out}}$ is a bias term.

2.1.2 Multi-Layer Perceptrons

Simple perceptrons are only capable of learning linear functions. We can nonetheless arrange multiple layers, with each layer i having its independent weight matrix $\boldsymbol{W^{i+1}} \in \mathbb{R}^{d_i \times d_{i+1}}$ and bias term $\boldsymbol{b^{i+1}} \in \mathbb{R}^{d_{i+1}}$ [Ivakhnenko and Lapa, 1973]. The Multi Layer Perceptron with one hidden layer (MLP1) is illustrated in Figure 2.2 and defined as:

$$\mathrm{NN}_{\mathrm{MLP1}}(\boldsymbol{x}) = \boldsymbol{W^2}\mathrm{g}(\boldsymbol{W^1}\boldsymbol{x} + \boldsymbol{b^1}) + \boldsymbol{b^2}. \tag{2.2}$$

More layers to the MLP can be added, and thus a MLPi can be defined as follows:

$$\mathrm{NN}_{\mathrm{MLP}i}(\boldsymbol{x}) = \begin{cases} \mathrm{NN}_{\mathrm{Perceptron}}(\boldsymbol{x}), & \text{if } i = 0 \\ \boldsymbol{W^{i+1}}\mathrm{g}(\mathrm{NN}_{\mathrm{MLP}i-1}(\boldsymbol{x})) + \boldsymbol{b^{i+1}}, & \text{if } i \geq 1. \end{cases} \tag{2.3}$$

In the previous equations, the function $g()$ is called a non-linearity or activation function, and it allows a MLP to approximate non-linear functions.

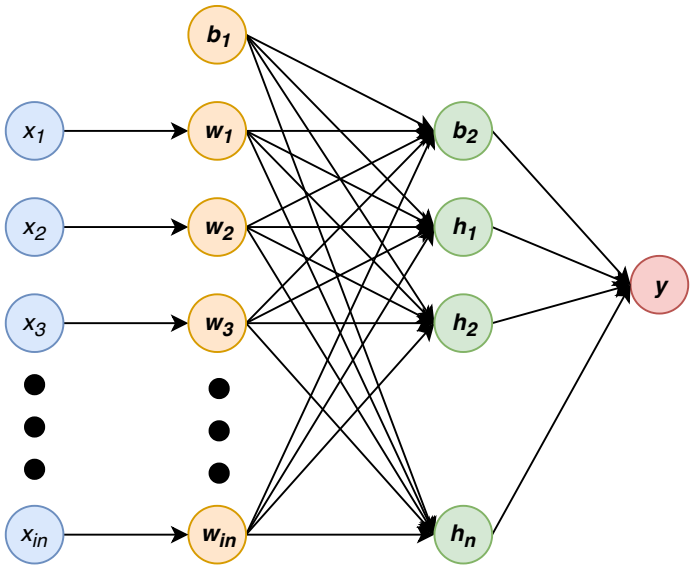


Figure 2.2: Illustration of a multi layer perceptron with one hidden layer. The green neurons are row vectors of hidden layers of the form $\boldsymbol{h} = \boldsymbol{W^2}\mathrm{g}(\boldsymbol{W^1}\boldsymbol{x} + \boldsymbol{b^1})$.

2.1.3 Recurrent Neural Networks

Traditional neural networks are limited to fixed size inputs. A Recurrent Neural Network (RNN) is, at a high level, a neural network where the connections between neurons are structured along a temporal sequence [Elman, 1990]. This allows the network to represent arbitrarily sized sequences, such as sentences in a document. Sequences can be defined through n d_{in}-dimensional vectors $\boldsymbol{x_{1:n}} = \boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}, (\boldsymbol{x_i} \in \mathbb{R}^{d_{in}})$. The output of an RNN is a single vector $\boldsymbol{y_n} \in \mathbb{R}^{d_{out}}$. For each prefix $\boldsymbol{x_{1:i}}$ of the sequence $\boldsymbol{x_{1:n}}$, an output vector $\boldsymbol{y_i} \in \mathbb{R}^{d_{out}}$ is implicitly defined by the previous equation. This output vector is a state of each stage of the RNN, which is represented by the function RNN*:

$$\begin{aligned} \boldsymbol{y_{1:n}} &= \mathrm{RNN}^*(\boldsymbol{x_{1:n}}) \\ \boldsymbol{y_i} &= \mathrm{RNN}(\boldsymbol{x_{1:i}}). \end{aligned} \tag{2.4}$$

The output vector $\boldsymbol{y_i}$ is then used for further prediction. The RNN function is defined recursively by means of a function $\mathrm{R}()$ that takes as input a state vector $\boldsymbol{s_{i-1}}$ and an input vector $\boldsymbol{x_i}$, returning a new state vector $\boldsymbol{s_i}$. This is then mapped to an output vector $\boldsymbol{h_i}$ using a simple deterministic function $\mathrm{O}()$. The dimensionality of the states $\boldsymbol{s_i}$ is a function of the output:

$$\begin{aligned} \mathrm{RNN}^*(\boldsymbol{x_{1:n}} : \boldsymbol{s_0}) &= \boldsymbol{y_{1:n}} \\ \boldsymbol{y_i} &= \mathrm{O}(\boldsymbol{s_i}) \\ \boldsymbol{s_i} &= \mathrm{R}(\boldsymbol{s_{i-1}}, \boldsymbol{x_i}). \end{aligned} \tag{2.5}$$

The Elman RNN [Elman, 1990], or Simple RNN (S-RNN), is a very simple RNN formulation that is sensitive to the ordering of elements in the input sequence. It linearly transforms each of the states $\boldsymbol{s_{i-1}}$ and the inputs $\boldsymbol{x_i}$, adds the results together with a bias term $\boldsymbol{b}$ and passes them through a nonlinearity $\mathrm{g}()$, as shown next and illustrated in Figure 2.3:

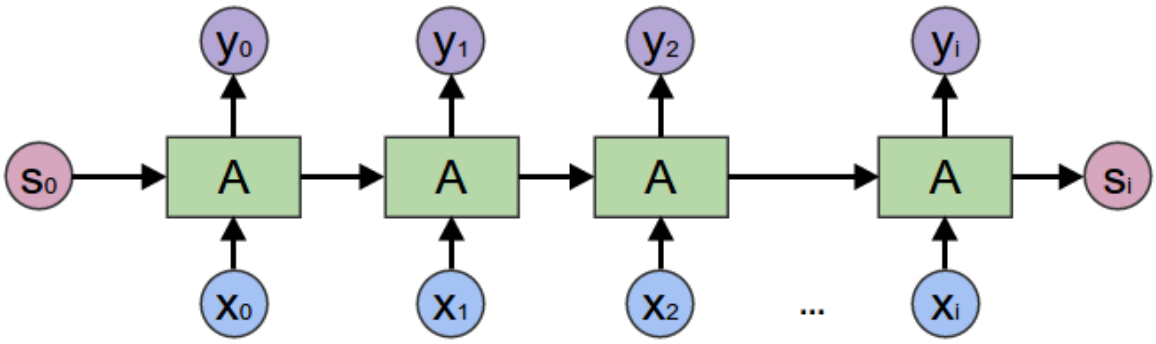


Figure 2.3: Illustration for an unrolled Elman RNN. The structure A corresponds to the function $\mathrm{R}()$ for generating states $\boldsymbol{s_i}$. Adapted from original image by Olah [2015a].

$$\begin{aligned} \boldsymbol{s_i} &= \mathrm{R}_{\mathrm{SRNN}}(\boldsymbol{x_i}, \boldsymbol{s_{i-1}}) = \boldsymbol{W}\mathrm{g}([\boldsymbol{s_{i-1}} : \boldsymbol{x_i}] + \boldsymbol{b}) \\ \boldsymbol{y_i} &= \mathrm{O}_{\mathrm{SRNN}}(\boldsymbol{s_i}) = \boldsymbol{s_i}. \end{aligned} \tag{2.6}$$

While the S-RNN is effective in acknowledging the ordering of elements in the input sequence, it is hard to train due to the vanishing gradients problem: error signals (gradients) in later steps in the sequence diminish quickly in the backpropagation process, and do not reach earlier input signals, making it hard for the S-RNN to capture long-range dependencies [Goldberg, 2017]. More details about the training of neural networks will be discussed in Section 2.1.4.

Gating-based architectures are designed to solve the aforementioned deficiencies. The function $\mathrm{R}()$ reads the current RNN state $\boldsymbol{s_i}$ and writes the result, resulting in a new state $\boldsymbol{s_{i+1}}$ at every step. In order to provide more controlled access to each state, we define a vector $\boldsymbol{g} \in \mathbb{R}^n$, and its values are passed through a sigmoid function $\sigma(\boldsymbol{g})$ to bind them to the range $[0, 1]$. Vector $\boldsymbol{g}$ acts as a gate to an n-dimensional vector $\boldsymbol{x}$, through the element-wise vector product $\boldsymbol{x} \odot \boldsymbol{g}$. Elements of $\boldsymbol{x}$ close to one are allowed to pass, while the ones close to zero are blocked. These values can be conditioned on the input and the current state, allowing the gating mechanism to be trained with a gradient-based approach. The Long Short Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997], illustrated in Figure 2.4, is one example of an RNN leveraging gating mechanisms. It explicitly splits the state vector $\boldsymbol{s_i}$ into two halves, where one half is treated as memory cells, which preserve the memory and the error gradients, and the other is working memory. For a timestep j, we define $\boldsymbol{c_j}$ as the memory component and $\boldsymbol{h_j}$ as the hidden state component. There are three gates, namely $\boldsymbol{i}$ (input), $\boldsymbol{f}$ (forget), and $\boldsymbol{o}$ (output). For each update of $\boldsymbol{c_j}$, $\boldsymbol{f}$ controls how much of the previous memory to keep, $\boldsymbol{i}$ controls how much of the proposed update $\boldsymbol{z}$ to keep, and $\boldsymbol{o}$ determines the value of $\boldsymbol{h_j}$ (which is also the output $\boldsymbol{y_j}$) based on $\boldsymbol{c_j}$, fed to the hyperbolic tangent function, as illustrated in Figure 2.4:

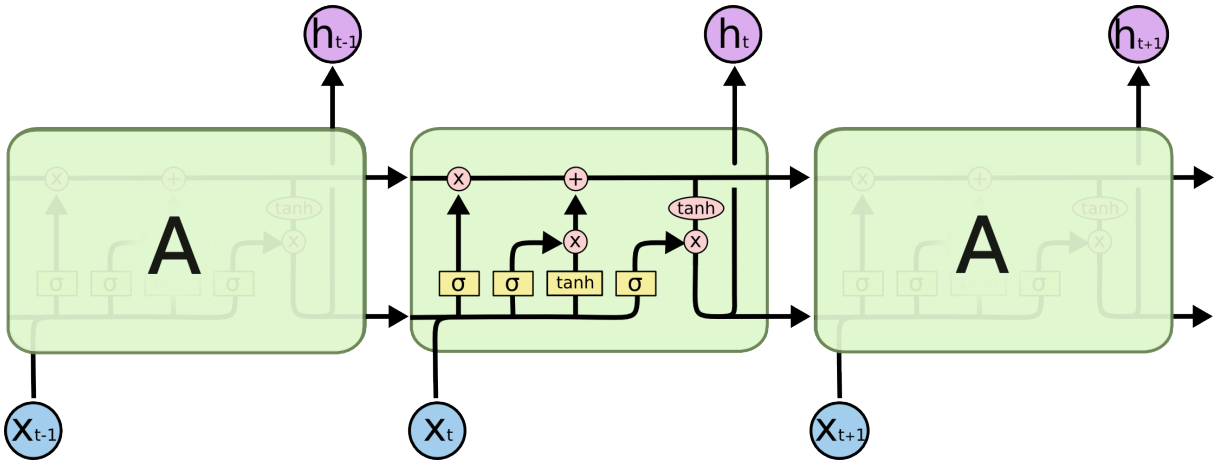


Figure 2.4: Illustration for the LSTM model, adapted from original image by Olah [2015b]

$$
\begin{aligned}
\boldsymbol{s_j} = \mathrm{R}_{\mathrm{LSTM}}(\boldsymbol{s_{j-1}}, \boldsymbol{x_j}) &= [\boldsymbol{c_j}; \boldsymbol{h_j}] \\
\boldsymbol{c_j} &= \boldsymbol{f} \odot \boldsymbol{c_{j-1}} + \boldsymbol{i} \odot \boldsymbol{z} \\
\boldsymbol{h_j} &= \boldsymbol{o} \odot \tanh(\boldsymbol{c_j}) \\
\boldsymbol{i} &= \sigma(\boldsymbol{W^{xi}} \boldsymbol{x_j} + \boldsymbol{W^{hi}} \boldsymbol{h_{j-1}}) \\
\boldsymbol{f} &= \sigma(\boldsymbol{W^{xf}} \boldsymbol{x_j} + \boldsymbol{W^{hf}} \boldsymbol{h_{j-1}}) \\
\boldsymbol{o} &= \sigma(\boldsymbol{W^{xo}} \boldsymbol{x_j} + \boldsymbol{W^{ho}} \boldsymbol{h_{j-1}}) \\
\boldsymbol{z} &= \tanh(\boldsymbol{W^{xz}} \boldsymbol{x_j} + \boldsymbol{W^{hz}} \boldsymbol{h_{j-1}}) \\
\boldsymbol{y_j} = \mathrm{O}_{\mathrm{LSTM}}(\boldsymbol{s_j}) &= \boldsymbol{h_j}.
\end{aligned}
\tag{2.7}
$$

2.1.4 Neural Network Training

The input to a supervised learning algorithm is a training set of n samples $\boldsymbol{x_{1:n}} = \boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}$, together with corresponding labels $\boldsymbol{y_{1:n}} = \boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_n}$. The output of one such algorithm is a function $\mathrm{f}()$ such that a loss function $\mathrm{L}(f(\boldsymbol{x}), \boldsymbol{y})$ is minimized. The function $\mathrm{L}()$ assigns a numerical score to a predicted output $\mathrm{f}(\boldsymbol{x})$ given the true expected output $\boldsymbol{y}$. The overall loss is defined as the total loss over all training examples, with respect to parameters Θ and a parameterized function $\mathrm{f}(\boldsymbol{x}; \Theta)$ [Goldberg, 2017]:

$$
\mathcal{L}(\Theta) = \sum_{i=1}^{n} \mathrm{L}(\mathrm{f}(\boldsymbol{x}_i; \Theta), \boldsymbol{y}_i). \tag{2.8}
$$

The goal of the algorithm is to set Θ such that the value of $\mathcal{L}()$ is minimized, usually also considering a regularization term $\mathrm{R}()$ to mitigate overfitting:

$$
\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}}\, \mathcal{L}(\Theta) + \lambda \mathrm{R}(\Theta). \tag{2.9}
$$

In the previous expression, λ is a scalar, the regularization rate.

Algorithm 1 Minibatch stochastic gradient descent training.

Input:
- Function $\mathrm{f}(\boldsymbol{x}; \Theta)$ parameterized with parameters Θ.
- Training set of inputs $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ and desired outputs $\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}$.
- Loss function $\mathrm{L}()$.
- Learning rate η.

while stopping criteria not met **do**
  Sample a minibatch of n examples $\{(\boldsymbol{x_1}, \boldsymbol{y_1}), \ldots, (\boldsymbol{x_m}, \boldsymbol{y_m})\}$
  $\hat{\boldsymbol{g}} \leftarrow 0$
  for $i = 1$ to n **do**
    Compute the loss $\mathrm{L}(\mathrm{f}(\boldsymbol{x}; \Theta), \boldsymbol{y_i})$ w.r.t Θ
    $\hat{\boldsymbol{g}} \leftarrow \hat{\boldsymbol{g}}$ + gradients of $\frac{1}{n}\mathrm{L}(\mathrm{f}(\boldsymbol{x}; \Theta), \boldsymbol{y_i})$ w.r.t Θ
  end for
  $\Theta \leftarrow \Theta - \eta\hat{\boldsymbol{g}}$
end while
return Θ

The loss function L() used for the training of neural networks is often the categorical cross entropy loss. Let $\boldsymbol{y} = \boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ be a vector representing the true multinomial distribution over the labels $1, \ldots, n$, and let $\hat{\boldsymbol{y}}_1, \ldots, \hat{\boldsymbol{y}}_n$ be the output of a final linear classifier, after a transformation through a softmax function, representing the class membership conditional distribution $\hat{\boldsymbol{y}}_i = P(y = i|\boldsymbol{x})$, where the vector $\boldsymbol{x}$ is the corresponding input sample. The categorical cross entropy loss measures the dissimilarity between the true label distribution $\boldsymbol{y}$ and the predicted label distribution $\hat{\boldsymbol{y}}$, and is defined as follows:

$$L_{\mathrm{cross-entropy}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = -\sum_i \boldsymbol{y}_i \log(\hat{\boldsymbol{y}}_i) \tag{2.10}$$

Neural networks are differentiable parameterized functions, and are trained using gradient-based optimization. An effective gradient method is Stochastic Gradient Descent (SGD) [Kiefer and Wolfowitz, 1952], which repeatedly computes an estimate of the loss $\mathcal{L}()$ over the training set, computing the gradients of the parameters Θ. To reduce eventual noise in the loss calculation, algorithm 1 estimates the error and the gradients based on a sample of m examples corresponding to a mini-batch. The rate at which parameters are updated is defined by a constant, scalar hyperparameter η. An efficient means to calculate gradients in the case of neural networks with multiple layers is the backpropagation algorithm, which methodically computes the derivatives of a nested composite expression using the chain-rule of differentiation, while caching intermediary results.

While SGD often produces good results for a large variety of tasks, extensions have been developed for the base SGD algorithm that have performed better in the training of neural networks [Goldberg, 2017]. One of such extensions is adaptive moment estimation (Adam) [Kingma and Ba, 2014, Loshchilov and Hutter, 2017], illustrated in algorithm 2, which foregoes a single, constant learning rate and instead maintains an adaptive learning rate that adjusts itself based on the values of the current gradients.

A neural network can be easily represented by means of the computation-graph abstraction. Nodes correspond to layers in the network and their parameters, while edges correspond to the flow of intermediary values between each layer. In a graph of N nodes, each node has an index i according to topological ordering. Let $\mathrm{f_i}()$ be the function computed by node i. Let also $\pi(i)$ be the ancestor nodes

Algorithm 2 Minibatch stochastic Adam training.

Input:
- Function $\mathrm{f}(\boldsymbol{x}; \Theta)$ parameterized with parameters Θ.
- Training set of inputs $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ and desired outputs $\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}$.
- Loss function $\mathrm{L}()$.
- Stepsize α.
- $\beta_1, \beta_2 \in [0, 1[$: Exponential decay rates for the moment estimates.
- $\epsilon \in \mathbb{R}$: A very small number to avoid division by zero.

while stopping criteria not met **do**
  Sample a minibatch of n examples $\{(\boldsymbol{x_1}, \boldsymbol{y_1}), \ldots, (\boldsymbol{x_m}, \boldsymbol{y_m})\}$
  $\boldsymbol{m} \leftarrow 0$
  $\boldsymbol{v} \leftarrow 0$
  for $i = 1$ to n **do**
    Compute the loss $\mathrm{L}(\mathrm{f}(\boldsymbol{x}; \Theta), \boldsymbol{y_i})$ w.r.t Θ
    $\boldsymbol{g} \leftarrow$ gradients of $\mathrm{L}(\mathrm{f}(\boldsymbol{x}; \Theta), \boldsymbol{y_i})$ w.r.t Θ
    $\boldsymbol{m} \leftarrow \beta_1 \cdot \boldsymbol{m} + (1 - \beta_1) \cdot \boldsymbol{g}$
    $\boldsymbol{v} \leftarrow \beta_2 \cdot \boldsymbol{v} + (1 - \beta_2) \cdot \boldsymbol{g^2}$
    $\hat{\boldsymbol{m}} \leftarrow \boldsymbol{m} / (1 - \beta_1^i)$
    $\hat{\boldsymbol{v}} \leftarrow \boldsymbol{v} / (1 - \beta_2^i)$
  end for
  $\Theta \leftarrow \Theta - \alpha \cdot \hat{\boldsymbol{m}} / (\sqrt{\hat{\boldsymbol{v}}} + \epsilon)$
end while
return Θ

Algorithm 3 Computation graph backward pass (backpropagation).

$d(N) \leftarrow 1$ ▷ $\frac{\partial N}{\partial N} = 1$
for $i = N - 1$ to 1 **do**
  $d(i) \leftarrow \sum_{j \in \pi(i)} d(j) \cdot \frac{\partial f_j}{\partial i}$ ▷ $\frac{\partial N}{\partial i} = \sum_{j \in \pi(i)} \frac{\partial N}{\partial j} \frac{\partial j}{\partial i}$
end for

of node i. For variable and input nodes, $\mathrm{f_i}()$ is a constant function. The backward pass begins by designating a node N with scalar output as a loss-node, and running forward computation up to that node. Denote by $\mathrm{d}(i)$ the quantity $\frac{\partial N}{\partial i}$. Algorithm 3 computes the values of $\mathrm{d}(i)$ for all nodes $i \in [1, N]$.

2.2 Representing Text for Machine Learning Algorithms

2.2.1 One-Hot Vector Representations

In one-hot vector representations, each feature (e.g., each word or word n-gram) is encoded in its own unique dimension of a representation vector $\boldsymbol{x}$. For example, if every word w in a vocabulary V is considered a feature, w is encoded as one specific dimension in a vector $\boldsymbol{x}$ of size $|V|$, with the value of one in the dimension corresponding to the word, and the value of zero elsewhere. To represent a document, this vector can be built from, usually, summation or averaging of all one-hot encodings corresponding to the words in the document. With this representation, the dimensionality of an one-hot vector is same as the number of distinct features, and features are completely independent from one another. Thus it is impossible to capture the order of the words in the sequence, and any meaning, contextual or not, is obscured [Goldberg, 2017].

2.2.2 Neural Language Models and Dense Vector Representations

Language modeling is the task of assigning a probability to sentences in a language, often being used for predicting which word follows a sequence of words [Bengio et al., 2006]. Formally, this corresponds to assigning a probability to any sequence of words $w_{1:n}$. A kth order Markov assumption considers that the next word in a sequence depends only on the last k words. Using the chain rule of probability, this can be written as:

$$P(w_{1:n}) \approx P(w_1|w_{1-k:0}) + \prod_{i=2}^{n} P(w_i|w_{i-k:i-1}). \tag{2.11}$$

For a neural network to perform this task, we can use a model where the input is a k-gram of words $w_{1:k}$ over a finite vocabulary V, and the output is a probability distribution over the next word. Each word w is associated with an embedding vector $\boldsymbol{v}(w) \in \mathbb{R}^{d_w}$, and an input vector $\boldsymbol{x} \in \mathbb{R}^{k \cdot d_w}$ to a prediction model can be built from a concatenation of the k words:

$$\boldsymbol{x} = [\boldsymbol{v}(w_1); \boldsymbol{v}(w_2); \ldots; \boldsymbol{v}(w_k)], w_i \in V.$$

The prediction model is defined as an MLP with one or more hidden layers, as shown next:

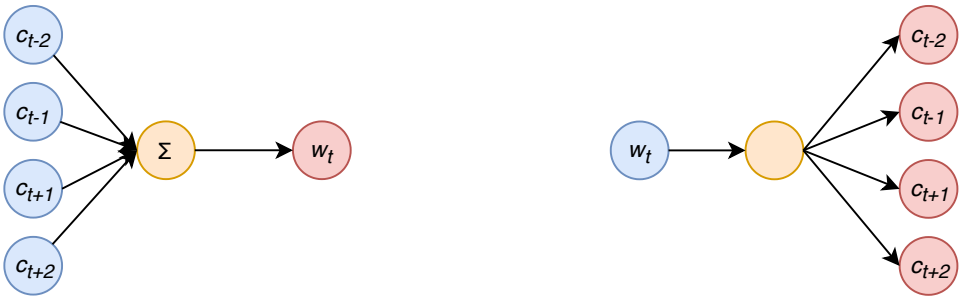
$$\begin{aligned} \hat{\boldsymbol{y}} = P(w_i|w_{1:k}) &= \mathrm{softmax}(\boldsymbol{W^2 h} + \boldsymbol{b^2}) \\ \boldsymbol{h} &= \mathrm{g}(\boldsymbol{W^1 x} + \boldsymbol{b^1}) \end{aligned} \tag{2.12}$$

The previous ideas regarding language modeling with neural networks have been used as the base approach to build more flexible text representation mechanisms, going beyond one-hot sparse vectors and representing individual words as dense vectors of lower dimensionality (i.e. embedding vectors). Dense vectors are less computationally heavy than one-hot encodings, and have more generalization power [Goldberg, 2017], as they are able to capture similarities between distinct features: learned vectors are similar if their corresponding words are seen in similar contexts.

Consider the neural language model in Equation (2.12). Each column of matrix $\boldsymbol{W^2}$ corresponds to a distributed representation of a word. The training process determines good values to the embeddings so that they produce correct probability estimates for a word in a certain context. Several previous studies have proposed different language modeling objectives to support the training of word representations. For instance, Word2Vec [Mikolov et al., 2013] attempts to train a network to find good word-context pairs (w, c). The evaluation of word-context pairs consists of computing a score $\mathrm{s}(w, c)$ for each word-context pair. Given a set D of correct word-context pairs generated from a given corpus, Word2Vec estimates the probability $P(D = 1|w, c)$ that the word-context pair came from the correct set D, given by:

$$P(D = 1|w, c) = \frac{1}{1 + e^{-\mathrm{s}(w,c)}}. \tag{2.13}$$

Let $\bar{D}$ be a set of *bad* word-context pairs. To maximize the log-likelihood of the data $D \cup \bar{D}$, the authors consider the following loss function:



(a) CBOW architecture.

(b) Skip-Gram architecture.

Figure 2.5: Illustration of the two scoring functions in Word2Vec.

$$\mathcal{L}(\Theta; D, \bar{D}) = \sum_{(w,c)\in D} \log P(D=1|w,c) + \sum_{(w,c)\in \bar{D}} \log P(D=0|w,c). \tag{2.14}$$

Word2Vec outputs a vector space in which word vectors are positioned such that words that share common contexts are located in close proximity to one another. The word-context score function $\mathrm{s}(w,c)$ has two implementations in the original Word2Vec proposal, as illustrated in Figure 2.5 and described as follows:

- *Continuous Bag of Words (CBOW)*: for a multi-word context $c_{1:k}$ the context vector $\boldsymbol{c}$ is defined to be a sum of the embedding vectors of the context components: $\sum_{i=1}^{k} \boldsymbol{c_i}$. The scoring function is then defined as simply $s(w,c) = \boldsymbol{w}\cdot\boldsymbol{c}$. Illustrated in Figure 2.5a, this approach predicts target words from all surrounding context words. By treating an entire context as one observation, it smoothes over most distributional information, which has shown to have better results for smaller corpora.

- *Skip-Gram*: for a multi-word context $c_{1:k}$, each element c_i in the context is assumed to be independent of each other. A word-context pair $(w, c_{i:k})$ is represented in D as k different contexts $(w,c_1),\ldots,(w,c_k)$. Therefore, each word-context pair is now its own embedding vector, and the scoring function is now defined as $s(w,c) = \boldsymbol{w}\cdot\boldsymbol{c_i}$, with $i \in [1..k]$. The resulting probability function is calculated by applying the product over all $\boldsymbol{c_i}$:

$$P(D=1|w,c_{1:k}) = \prod_{i=1}^{k} P(D=1|w,c_i) = \prod_{i=1}^{k} \frac{1}{1+e^{-\boldsymbol{w}\cdot\boldsymbol{c_i}}} \tag{2.15}$$

  Illustrated in Figure 2.5b, this approach predicts surrounding context words from the target words, by treating each target-context pair as a new observation, which was shown to have better results for larger corpora.

2.2.3 Contextual Distributed Embeddings

Algorithms like Word2Vec already take into account the different contexts where a word appears in during the training process. However, the representation for a word is the same for all contexts a word may appear in. This is particularly limiting, as many words have different meanings depending on the

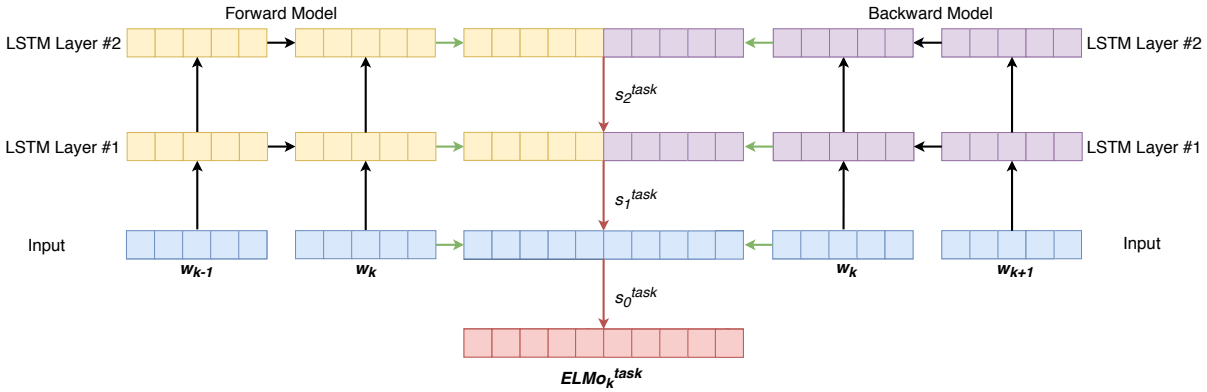


Figure 2.6: Illustration for the ELMo architecture with two LSTM layers. Black arrows signify that the origin is used as input to the target, green arrows represent vector concatenation, and red arrows represent weighted vector sum.

context. Contextualized representations attempt to model the characteristics of word usage as well how this usage varies across different contexts. One particular implementation of these principles is Embeddings from Language Models (ELMo) [Peters et al., 2018], illustrated in Figure 2.6. ELMo uses vectors derived from a bidirectional LSTM that is trained with a coupled language modeling objective. A forward language model, as defined in Equation (2.11), computes the probability of a sequence of words given the sequence of words that precedes it, while a backward language model computes the probability of a sequence of words given the words that come after it:

$$P(w_{1:n}) \approx \prod_{i=1}^{n} P(w_i | w_{i+1:n})$$

A bidirectional language model combines these two approaches, for instance maximizing the log likelihood of the forward and backward directions:

$$\begin{aligned} \sum_{i=1}^{n} (&\log P(w_i | w_{1:i-1}; \Theta_x, \overrightarrow{\Theta}_{LSTM}, \Theta_s) \\ &+ \log P(w_i | w_{i+1:n}; \Theta_x, \overleftarrow{\Theta}_{LSTM}, \Theta_s)). \end{aligned} \tag{2.16}$$

In the previous expression, Θ_x, Θ_{LSTM}, and Θ_s are parameters for the word embeddings, for each LSTM, and for a final softmax layer, respectively. The resulting model is a stacked, multi-layer LSTM, where each layer takes the previous layer's output as input. For a bidirectional LSTM of L layers, ELMo is able to produce $2 \cdot L + 1$ different representations for a given word, including the original context-independent representation. A task-specific embedding vector $\boldsymbol{ELMo}_k^{task}$ for a word of index k is constructed as represented in the following equation:

$$\boldsymbol{ELMo}_k^{task} = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \boldsymbol{h}_{kj} \tag{2.17}$$

In the previous equation, the index j is the index of the layer the embedding is being extracted from and $\boldsymbol{h}_{kj}$ is the hidden state output of the layer of index j for the word of index k. s_j^{task} is a task-specific learned weight for each hidden state, while the parameter γ^{task} is a task-specific value that allows for

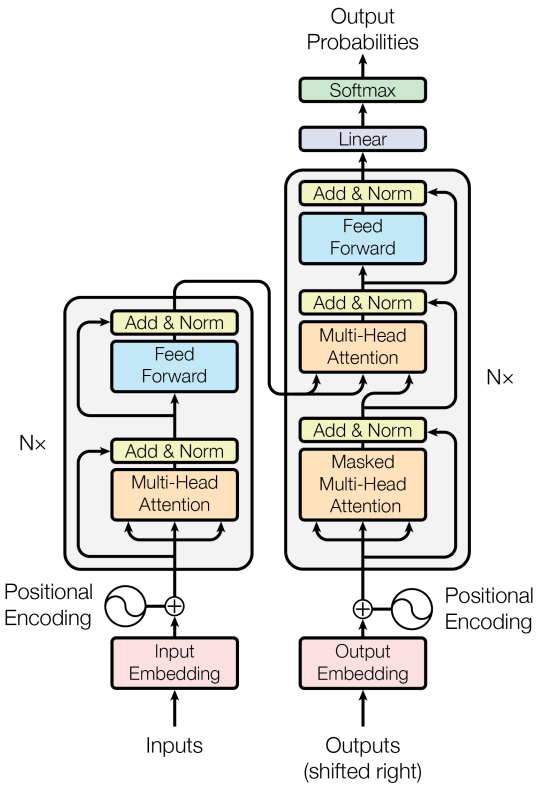


Figure 2.7: Illustration for the Transformer model, adapted from Vaswani et al. [2017].

scaling the entire vector, which is important during optimization. The final embeddings are obtained by computing a weighted sum of these hidden states.

2.2.4 Transformer-based Architectures for Contextual Representations

The Transformer [Vaswani et al., 2017] moves away from recurrent architectures, replacing sequential mechanisms with attention. This allows a model to process the entirety of a sequence in one step, instead of sequentially from left-to-right and/or right-to-left. The overall architecture of the Transformer is illustrated in Figure 2.7. It consists of an encoder, which takes a set of word embeddings and produces a representation of that input, and a decoder that takes this representation and produces a prediction for the task at hand (e.g., language modeling).

The encoder and decoder structures employ attention mechanisms, which allow the decoder to look back at the entire sentence and selectively extract the information needed during decoding, effectively preserving long-term dependencies. Attention is defined as the relevance of a set of values $\boldsymbol{V}$ based on some keys $\boldsymbol{K}$ and queries $\boldsymbol{Q}$. Each query $\boldsymbol{q} \in \boldsymbol{Q}$ and key $\boldsymbol{k} \in \boldsymbol{K}$ are of dimensionality d_k, while each value $\boldsymbol{v} \in \boldsymbol{V}$ is of dimensionality d_v. The output is a weighted sum of of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

A compatibility function assigns a score to each pair of words indicating how strongly they should attend to one another. For each word in the input sequence, the Transformer first assigns a query vector $\boldsymbol{q}$, which corresponds to the encoder hidden state for the word that is paying attention, i.e. the one that is *querying* the other words, and a key vector $\boldsymbol{k}$, which corresponds to the encoder hidden state for the

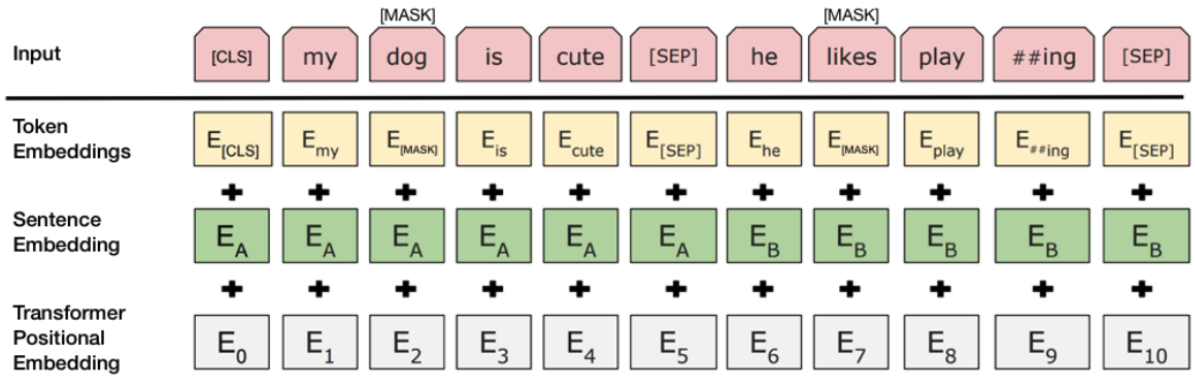


Figure 2.8: Illustration for BERT embeddings, adapted from Devlin et al. [2019].

word to which attention is being paid. The value v corresponds to the encoder hidden state for a single word in the sequence. In particular, the Transformer uses a specific form of attention named scaled dot product attention, which feeds the dot product of all keys and queries through a softmax function, multiplying the result with all values:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\intercal}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{2.18}$$

Multi-head attention consists of linearly projecting the queries, keys and values multiple times with different, learned linear projections to d_k, d_k and d_v dimensions, respectively. This allows the model to jointly attend to information from different representation subspaces at different positions.

BERT

Bidirectional Encoder Representations from Transformers (BERT) [Devlin et al., 2019], illustrated in Figure 2.8, takes advantage of the encoder part of the Transformer to produce word embeddings. Like in ELMo, the representation of a given word is a function of the entire input sequence, composed of three different embeddings: a word embedding, for a single word in the input sequence; a sentence embedding, for the sentence that the word belongs to, and a positional embedding, for the position of the word in the input sequence. The latter is directly taken from the Transformer encoder, and it is necessary for the model to make use of the order of the sequence, as it has no recurrence.

For word embeddings, BERT masks some percentage of the input words at random, and then predicts those masked words, ignoring the prediction of the non-masked words. This is called a Masked Language Modeling (MLM) objective, which keeps the Transformer encoder from knowing which words it will be asked to predict, so it is forced to keep a distributional contextual representation of every input token. For sentence classification, BERT is trained on the task of Next Sentence Prediction (NSP), in which the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. The first token of every sequence is always a special classification token [CLS], and the two sentences are separated with the token [SEP]. The [CLS] token is a special token used to incorporate features that are useful for classification, as it attends to all tokens in the sequence. The final embedding is computed with an identification for each token in

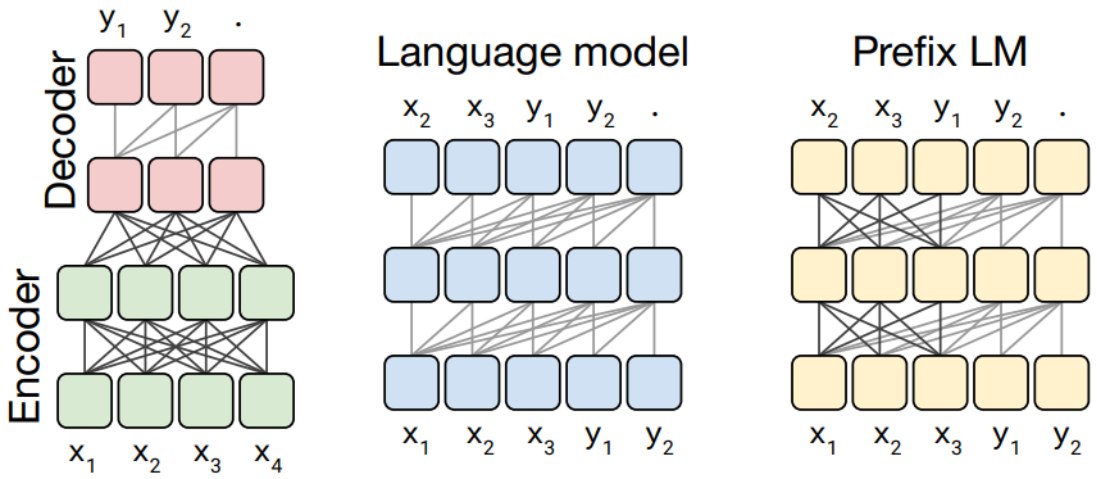


Figure 2.9: Illustration of the architectures considered by the T5 authors, adapted from [Raffel et al., 2020]. Blocks represent elements of a sequence and lines represent attention visibility. Different colored groups of blocks indicate different Transformer layer stacks. Dark grey lines correspond to fully-visible masking and light grey lines correspond to causal masking. "." denotes the end of a prediction.

the sequence, as belonging to sentence A or sentence B. MLM and NSP are trained together, with the goal of minimizing the combined loss function of the two strategies. BERT is capable of incorporating large amounts of knowledge from being pre-trained in large corpora. This causes the model to develop general-purpose abilities and knowledge that can then ideally be transferred to more specific tasks.

Thus, the most important feature BERT has introduced to the domain of NLP is its ability to be fine-tuned on downstream tasks. After a pre-training step on the aforementioned MLM and NSP tasks, all of the parameters that have been learned from pre-training can fine-tuned in a supervised context using labeled data from a specific downstream task, including but not limited to: machine translation, text summarization, and question answering. An example of a study that explore how BERT and other Transformer-based models can be fine-tuned for text summarization are described in section 2.4.1.

T5

Following BERT's success in the domain of text representation, other language models have been built upon it to explore the Transformer's capabilities in a variety of downstream tasks. The Text-To-Text Transfer Transformer (T5) [Raffel et al., 2020] was introduced as a survey of modern transfer learning techniques used in NLP, proposing a unified framework that combines all language tasks into a text-to-text format. This allows us to fine-tune a single model on several different downstream tasks, maintaining the same hyperparameters and training objective across each context. This is achieved by adding a task-specific prefix to the original input sample, before passing it to the model. For example, "summarize:" and "translate English to German:" are prepended to the input sequence to inform the model that the expected outcome is for a summarization and English-German translation tasks, respectively.

To achieve this, the authors experimented with two different applications for the attention mechanism: fully-visible attention masking, as used by BERT on its [CLS] token, allows a self-attention mechanism to attend to all entries of the input sequence when generating entries for the output sequence. Causal

attention masking only allows a self-attention mechanism to attend to entries that come before it. This is the strategy used in the Transformer decoder, so that the model can't "see into the future" as it produces its output. This was further studied in the context of three different architectures, illustrated in figure 2.9:

- The original Transformer encoder-decoder architecture. This architecture corresponds to using fully-visible attention masking in the encoder to produce a representation, and casual attention masking in the decoder for generating the output.
- A language modeling architecture, which uses a single Transformer layer stack and is fed the concatenation of the input and target, using a causal attention mask throughout. This means that the model's representation of a given entry only depends on the tokens before it.
- A prefixed language model maintains the general language model architecture, but replaces the causal attention masking over the prefix with fully-visible attention masking. The entries that do not belong to the prefix keep the causal attention masking.

In the author's experiments, the encoder-decoder architecture yielded the best results, despite incurring more computational cost due to the increased number of parameters. The architecture of the resulting model is, as such, very similar to the basic encoder-decoder Transformer, the most notable difference being the usage of relative positional embeddings instead of a fixed embedding for each position. Embeddings are instead generated from the offset between the *key* and *query* being compared in the Transformer's attention mechanism. A similar pre-training goal to BERT is used: a denoising objective where 15% of the words are masked and the model is trained to predict them.

2.3 Evaluation Metrics for Text Summarization

Among the first strategies developed for the evaluation of for summarization models were manual and semi-automatic metrics based on the pyramid approach [Nenkova and Passonneau, 2004, Passonneau et al., 2013]. These strategies, although robust, are completely dependent on human judgement, and as such it is very costly and cumbersome to apply them to large-scale summarization models [Kryscinski et al., 2019]. Therefore, much effort has been made to develop fast automatic metrics for evaluation. This section explores evaluation metrics that have been proposed in the literature to automatically evaluate large-scale summarization models.

2.3.1 ROUGE

The most typical automatic metric for the evaluation of automatically generated summaries is the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) package [Lin, 2004]. ROUGE measures lexical overlap between the generated summaries and a set of reference summaries, usually human-written. ROUGE scores are based on exact token matches, meaning it does not consider overlap between synonymous phrases. Overlap is computed between consecutive sub-sequences of tokens.

It provides a set of score functions that measure recall and precision, named ROUGE-N, which measures n-gram overlap; ROUGE-L, which measures the longest matching sequence of words using longest common subsequence (LCS) matching, and ROUGE-S, which measures the overlap of skip-bigrams, which are word pairs that can have a maximum of two gaps in-between words. The recall, precision and F1 score for ROUGE-N are calculated as follows:

$$R_{\mathrm{ROUGE-N}} = \frac{\sum_{(gen,ref)\in S} \sum_{gram_n \in ref} Count_{gen}\left(gram_n\right)}{\sum_{(gen,ref)\in S} \sum_{gram_n \in ref} Count_{ref}\left(gram_n\right)} \tag{2.19}$$

$$P_{\mathrm{ROUGE-N}} = \frac{\sum_{(gen,ref)\in S} \sum_{gram_n \in ref} Count_{gen}\left(gram_n\right)}{\sum_{(gen,ref)\in S} \sum_{gram_n \in gen} Count_{gen}\left(gram_n\right)} \tag{2.20}$$

$$F1_{\mathrm{ROUGE-N}} = \frac{2 \cdot R_{\mathrm{ROUGE-N}} \cdot P_{\mathrm{ROUGE-N}}}{R_{\mathrm{ROUGE-N}} + P_{\mathrm{ROUGE-N}}} \tag{2.21}$$

In the previous expressions, gen, ref and S refer to a generated summary, its corresponding reference summary and the set of summaries under study, respectively. $gram_n$ is an n-gram phrase and $Count_{gen}\left(gram_n\right)$ and $Count_{ref}\left(gram_n\right)$ refer to the occurrence number of $gram_n$ in gen and ref, respectively. The expressions for ROUGE-L and ROUGE-S are similar to the ones presented, with the counts for overlapping n-grams being replaced by the length of the LCS and counts for overlapping skip-bigrams, respectively. When comparing two approaches, the F1 scores are given more consideration, as they measure a balance of precision and recall. As noted by the authors, ROUGE scores correlate better with human judgement if generated from multiple references.

ROUGE is simple and its scores easily interpretable. However, there are several problems associated with its usage for the evaluation of summarization models, stemming from its inability to consider valid overlap between synonymous phrases [Kryscinski et al., 2019, Fabbri et al., 2020]. Most notably:

1. It assesses only content selection, and as such provides only a measure of adequacy for the summary; it does not account for other quality aspects, such as fluency and coherence.

2. In abstractive summarization, there are many ways to represent the same factual knowledge as in the reference summary without using the same n-grams, making a metric based on this principle unsuitable for this task.

3. Summarization is a subjective task, and subsequently ROUGE is designed to account for multiple references per input, with generated summaries being rewarded for sharing n-grams with more than one reference. However, most datasets provide only one reference.

4. As noted by Cohan and Goharian [2016], the high correlation with human judgments shown by the ROUGE authors is based on experiments with news data, which is intrinsically very different than other summarization tasks, such as the summarization of online discussions or scientific papers.

For these reasons, basing results solely on ROUGE scores is seldom sufficient for claiming state-of-the-art. Most studies carry out additional manual comparisons of alternative summaries. As this

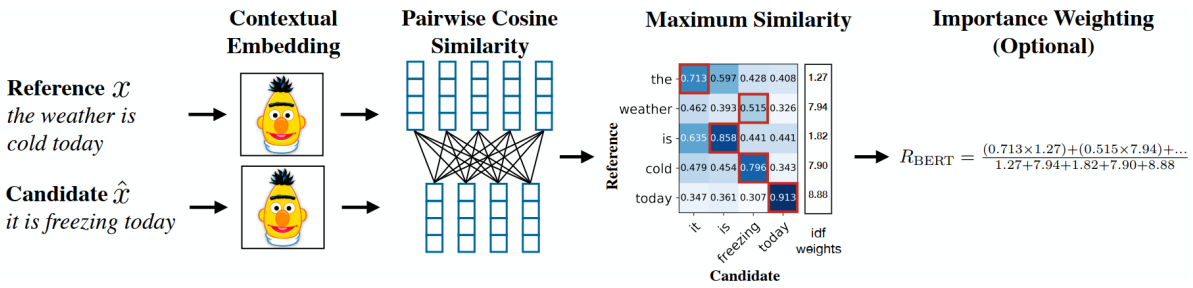


Figure 2.10: Illustration of the computation of the recall metric in BERTScore, adapted from Zhang et al. [2020].

is difficult to compare across studies, ROUGE scores are most useful when paired with more flexible metrics that are able to consider synonymous phrases in addition to n-gram overlap, and thus correlate better with human judgement.

2.3.2 ROUGE-WE

ROUGE-WE [Ng and Abrecht, 2015] is one of the first attempts to extend ROUGE with support for synonyms and paraphrases of the reference summaries, by replacing exact lexical matches with a soft semantic similarity measure, approximated with the cosine distances between word embeddings. Thus, a similarity function f_{WE} measuring similarity between two words w_1 and w_2 is defined as follows:

$$f_{WE} = \begin{cases} 0, & \text{if } \boldsymbol{v_1} \text{ or } \boldsymbol{v_2} \text{ are OOV} \\ \boldsymbol{v_1} \cdot \boldsymbol{v_2}, & \text{otherwise} \end{cases} \tag{2.22}$$

In the previous equation, $\boldsymbol{v_x}$ is the word embedding for word w_x. The embeddings used for computing the soft lexical matching are generated through Word2Vec.

2.3.3 BERTScore

BERTScore [Zhang et al., 2020] further enhances the soft similarity measure introduced in ROUGE-WE by introducing contextual embeddings generated through BERT. As ROUGE-WE generates embeddings through Word2Vec, it fails to account for the contextual meaning of words in a sentence. Given a sequence of reference vectors $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k]$ and a sequence of candidate vectors $\hat{\boldsymbol{X}} = [\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_m]$, of lengths k and m respectively, the recall, precision, and F1 scores for BERTScore are given by the following expressions:

$$R_{\text{BERT}} = \frac{1}{|\boldsymbol{x}|} \sum_{\boldsymbol{x}_i \in \boldsymbol{x}} \max_{\hat{\boldsymbol{x}}_j \in \hat{\boldsymbol{x}}} \boldsymbol{x}_i^\intercal \hat{\boldsymbol{x}}_j \tag{2.23}$$

$$P_{\text{BERT}} = \frac{1}{|\hat{\boldsymbol{x}}|} \sum_{\hat{\boldsymbol{x}}_j \in \hat{\boldsymbol{x}}} \max_{\boldsymbol{x}_i \in \boldsymbol{x}} \boldsymbol{x}_i^\intercal \hat{\boldsymbol{x}}_j \tag{2.24}$$

$$F_{\text{BERT}} = 2\frac{P_{\text{BERT}} \cdot R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}} \tag{2.25}$$

In the previous expressions, $\boldsymbol{x}^\intercal \hat{\boldsymbol{x}}$ is the cosine similarity between vectors $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$, assuming these vectors are normalized. In addition, BERTScore allows to optionally incorporate Inverse Document Frequency (IDF) [Jones, 1972] into the final scores. Given M reference sentences $\{\boldsymbol{x}^{(i)}\}_{i=1}^{M}$, the IDF score of a word-piece token $\boldsymbol{w}$ is described by the following expression:

$$\mathrm{IDF}(w) = -\log \frac{1}{M} \sum_{i=1}^{M} \mathcal{I}[w \in \boldsymbol{x}^{(i)}] \tag{2.26}$$

In the previous equation, $\mathcal{I}[\cdot]$ is an indicator function. The computation for the recall metric with IDF weighting is described as follows:

$$R_{\mathrm{BERT}} = \frac{\sum_{\boldsymbol{x_i} \in \boldsymbol{x}} \mathrm{IDF}(\boldsymbol{x_i}) \max_{\hat{\boldsymbol{x}}_j \in \hat{\boldsymbol{x}}} \boldsymbol{x_i}^\intercal}{\sum_{\boldsymbol{x_i} \in \boldsymbol{x}}} \tag{2.27}$$

The formulation for precision with IDF weighting is similar to the one described above for recall. Because BERTScore uses reference sentences to compute IDF, the IDF scores remain the same for all systems evaluated on a specific test set. These operations are illustrated in figure 2.10 for recall.

Finally, the authors note that, in practice, scores are observed in a limited range, potentially because of the learned geometry of contextual embeddings. This characteristic does not impact BERTScore's evaluation abilities; however, the numerical score is less readable. This is addressed by linearly rescaling BERTScore with respect to an empirical lower bound b as a baseline. The baseline b is computed by averaging BERTScore computed on sentence pairs randomly sampled from the Common Crawl dataset[1]. For example, the rescaled version of the BERTScore F-score is described as follows:

$$F_{\mathrm{BERT}} = \frac{F_{\mathrm{BERT}} - b}{1 - b} \tag{2.28}$$

Despite being a step in the direction of a more reliable evaluation strategy, metrics like ROUGE-WE and BERTScore have not gained sufficient popularity at the time of writing, leaving ROUGE as the default automatic evaluation toolkit for text summarization [Kryscinski et al., 2019].

2.4 Related Work

In this section, we present a detailed review of recent works pertaining to the task of text summarization using deep learning techniques for natural language processing in section 2.4.1, followed by relevant applications of NLP to the domain of online discussions in section 2.4.2.

2.4.1 Natural Language Processing in Text Summarization

Deep learning methods for natural language processing have been important in defining effective mechanisms for tasks related to language understanding and generation, with a particular example being the task of text summarization. We present and critically analyze different approaches to this task. In particular, many recent approaches have been extensively compared and evaluated by Kryscinski et al.

[1] `https://commoncrawl.org/`

[2019]. Three different types of approaches have been taken, namely extractive, abstractive, and hybrid summarization. The latter is a combination of extractive and abstractive approaches.

Extractive summarization directly transfers spans of text from the input to produce a summary. Extractive summaries will necessarily comply to baseline levels of grammatical accuracy. However, these approaches are incapable of paraphrasing, generalization, or incorporating other real-world knowledge [See et al., 2017]. For instance, Nallapati et al. [2016b] have introduced two approaches for extractive summarization, which we describe next. In both approaches, a set of word-level Gated Recurrent Unit (GRU) [Cho et al., 2014] based RNNs is run independently over each sentence in the document, where each time-step of the RNN corresponds to a word index in the sentence. The hidden states $\boldsymbol{h}$ of these bidirectional RNNs are then used an input to another bidirectional RNN whose time steps correspond to sentence indices in the document, and that generate sentence representations. The average pooling of $\boldsymbol{h}$ is used as the document representation $\boldsymbol{d}$. In addition, a dynamic summary representation $\boldsymbol{s}$ is defined, whose estimation is architecture dependent. The aforementioned ideas can be used to define a score function $\mathrm{s}(\boldsymbol{h_j}, \boldsymbol{s_j}, \boldsymbol{d}, \boldsymbol{p_j})$, where j is the index of the sentence in the document and $\boldsymbol{p_j}$ is the positional embedding of the sentence:

$$\begin{aligned}
\mathrm{s}(\boldsymbol{h_j}, \boldsymbol{s_j}, \boldsymbol{d}, \boldsymbol{p_j}) = w_c \cdot \sigma(\boldsymbol{W_c}^{\top} \cdot \boldsymbol{h_j}) \quad & \texttt{\#(content richness)} \\
+ w_s \cdot \sigma(\cos(\boldsymbol{h_j}, \boldsymbol{d})) \quad & \texttt{\#(salience w.r.t. document)} \\
+ w_p \cdot \sigma(\boldsymbol{W_p}^{\intercal} \cdot \boldsymbol{p_j}) \quad & \texttt{\#(positional importance)} \\
- w_r \cdot \sigma(\cos(\boldsymbol{h_j}, \boldsymbol{s_j})) \quad & \texttt{\#(redundancy w.r.t. summary)} \\
+ b), \quad & \texttt{\#(bias term)}
\end{aligned} \tag{2.29}$$

In the previous equation, $\boldsymbol{W}_c$ and $\boldsymbol{W}_p$ are parameter matrices to model content richness and positional importance of sentences respectively. The parameters w_c, w_s, w_p and w_r are learned scalar weights that model abstract features.

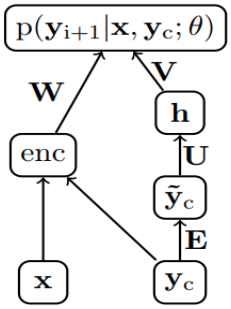
The first approach is a classifier architecture which scans the entire document sentence by sentence and classifies each sentence as being fit to be included in the summary. The probability that the sentence belongs to the summary, $P(y_j = 1)$, is given as follows from the score function.

$$P(y_j = 1 | \boldsymbol{h_j}, \boldsymbol{s_j}, \boldsymbol{d}, \boldsymbol{p_j}) = \sigma(\mathrm{s}(\boldsymbol{h_j}, \boldsymbol{s_j}, \boldsymbol{d}, \boldsymbol{p_j})) \tag{2.30}$$

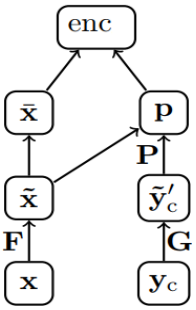
In this case, training is performed by minimizing the negative log-likelihood of the training data labels.

The second approach is a selector architecture which picks one sentence at a time in the order that it sees fit. In this architecture, the act of picking a sentence is cast as a sequential generative model in which one sentence-index is emitted at each time-step, selecting that which maximizes the score in Equation (2.29). As such, the probability of picking a sentence with index $I(j) = k \in \{1, \ldots, N_d\}$, where N_d is the size of the document in terms of the number of sentences, at time-step j, is given by the softmax over the scoring function:

$$P(I(j) = k | \boldsymbol{s_j}, \boldsymbol{h_k}, \boldsymbol{d}) = \mathrm{softmax}(\mathrm{s}(\boldsymbol{h_j}, \boldsymbol{s_j}, \boldsymbol{d}, \boldsymbol{p_j})). \tag{2.31}$$



(a) Decoder architecture.



(b) Encoder architecture.

Figure 2.11: Illustration for the basic encoder-decoder architecture for sequence-to-sequence transduction, adapted from Rush et al. [2015]

In this case, training is performed by minimizing the negative log-likelihood of the N_d selected sentences in the ground truth data. When generating summaries, the index of each sentence is picked iteratively at each time-step j, using the best score given the current summary representation:

$$I(j) = \underset{k \in \{1,\ldots,N_d\}}{\operatorname{argmax}} \ \mathrm{s}(\boldsymbol{h_j}, \boldsymbol{s_j}, \boldsymbol{d}, \boldsymbol{p_j}) \tag{2.32}$$

Abstractive summarization generates summaries containing sentences that were not necessarily present in the original text. Spans of text are instead paraphrased to form the final output. The goal is to select the best sequence $\boldsymbol{y}$ from a set of sentences $\mathcal{Y}$ that can be generated from the input $\boldsymbol{x}$, according to a scoring function $\mathrm{s}(\boldsymbol{x}, \boldsymbol{y})$.

Rush et al. [2015] define $\mathrm{s}()$ as a factored scoring function that takes into account a fixed window of previous words, calculating the conditional log-probability of a summary given the input:

$$\mathrm{s}(\boldsymbol{x}, \boldsymbol{y}) = \log p(\boldsymbol{y}|\boldsymbol{x}; \Theta) \approx \sum_{i=0}^{N-1} \log p(\boldsymbol{y}_{i+1}, \boldsymbol{x}, \boldsymbol{y}_c). \tag{2.33}$$

In the previous expression, $\boldsymbol{y}_c$ is defined as $\boldsymbol{y}_{[i-C+1,\ldots,i]}$ for a window of size C. This leads into a definition of an encoder-decoder architecture for the task of sentence-level summarization, for instance leveraging attention-based abstractive neural language models. Taking this scoring function into account, the goal is to model a local conditional distribution over the window. Take the parameters $\boldsymbol{E} \in \mathbb{R}^{d_w \times V}$ as a word embedding matrix and $\boldsymbol{U} \in \mathbb{R}^{(C \cdot d_w) \times d_h}$, $\boldsymbol{V} \in \mathbb{R}^{V \times d_h}$, and $\boldsymbol{W} \in \mathbb{R}^{V \times d_h}$ as weight matrices, where $\boldsymbol{w}$ is a word embedding and $\boldsymbol{h}$ is a hidden layer, as well as an additional encoder $\mathrm{enc}(\boldsymbol{x}, \boldsymbol{y}_c)$ built from Bengio et al. [2003]. The decoder, illustrated in Figure 2.11a, is defined as:

$$\begin{aligned} p(\boldsymbol{y}_{i+1}|\boldsymbol{y}_c, \boldsymbol{x}; \Theta) &\propto \exp(\boldsymbol{V} \cdot \boldsymbol{h} + \boldsymbol{W} \cdot \mathrm{enc}(\boldsymbol{x}, \boldsymbol{y}_c)), \\ \tilde{\boldsymbol{y}}_c &= [\boldsymbol{E} \cdot \boldsymbol{y}_{i-C+1}, \ldots, \boldsymbol{E} \cdot \boldsymbol{y}_i], \\ \boldsymbol{h} &= \tanh(\boldsymbol{U} \cdot \tilde{\boldsymbol{y}}_c). \end{aligned} \tag{2.34}$$

The encoder model $\mathrm{enc}(\boldsymbol{x}, \boldsymbol{y}_c)$, illustrated in Figure 2.11b, acts as a conditional generation model, originally defined by Bahdanau et al. [2014] as a translation model:

$$
\begin{aligned}
\text{enc}(\boldsymbol{x}, \boldsymbol{y}_c) &= \boldsymbol{p}^\intercal \overline{\boldsymbol{x}}, \\
\boldsymbol{p} &\propto \exp(\tilde{\boldsymbol{x}} \boldsymbol{P} \tilde{\boldsymbol{y}'}_c), \\
\tilde{\boldsymbol{x}} &= [\boldsymbol{F}\boldsymbol{x}_1, \ldots, \boldsymbol{F}\boldsymbol{x}_M], \\
\tilde{\boldsymbol{y}'}_c &= [\boldsymbol{G}\boldsymbol{y}_{i-C+1}, \ldots, \boldsymbol{G}\boldsymbol{y}_i], \\
\forall i \; \overline{\boldsymbol{x}}_{\boldsymbol{i}} &= \sum_{q=i-Q}^{i+Q} \tilde{\boldsymbol{x}}_i / Q,
\end{aligned}
\tag{2.35}
$$

In the previous expression, $\boldsymbol{F}$ is a word embedding matrix, $\boldsymbol{G} \in \mathbb{R}^{d_w \times V}$ is an embedding of the context, $\boldsymbol{P} \in \mathbb{R}^{d_h \times (C \cdot d_w)}$ is a new weight matrix parameter mapping between the context embedding and input embedding, M is the number of words, and Q is a value that defines the size of the set of values considered in the last sum in the equation, called a smoothing window. When generating summaries, the authors employ the usage of the beam-search algorithm, modified for the feed-forward model. Beam search employs a semi-greedy approach to finding the words to generate sentences in the summary, considering $\mathcal{K}$ multiple hypothesis in each generation step.

This model has demonstrated good results on small input and output sequences. However, it struggles to maintain coherence when applied on larger documents. Summaries are prone to repetition, and it does not take into account out-of-vocabulary (OOV) words. To mitigate this problem, See et al. [2017] develop a hybrid model named pointer-generator network, which combines the ease and accuracy of extractive methods with the ability to generate new words characteristic of abstractive methods.

To deal with content repetition, the authors apply a coverage mechanism. The coverage vector $\boldsymbol{c_t}$ represents the parts of the original text that have been considered by the attention mechanism:

$$
\begin{aligned}
\boldsymbol{e_i^t} &= \boldsymbol{v}^\intercal \tanh(\boldsymbol{W_h} \cdot \boldsymbol{h_i}) + \boldsymbol{W_s} \cdot \boldsymbol{s_t} + \boldsymbol{w_c} \cdot \boldsymbol{c_i^t} + \boldsymbol{b_{\text{attn}}} \\
\boldsymbol{a^t} &= \text{softmax}(\boldsymbol{e^t}).
\end{aligned}
\tag{2.36}
$$

In the previous expression, $\boldsymbol{h_i}$ are encoder hidden states, $\boldsymbol{s_t}$ are decoder states for every step t, and $\boldsymbol{v}$, $\boldsymbol{W_h}$, $\boldsymbol{W_s}$, $\boldsymbol{w_c}$ and $\boldsymbol{b_{\text{attn}}}$ are learned parameters, with $\boldsymbol{w_c}$ having the same length as $\boldsymbol{v}$. Next, the attention distribution is used to produce a weighted sum of the encoder hidden states, known as the context vector $\boldsymbol{h_t^*}$:

$$
\boldsymbol{h_t^*} = \sum_i \boldsymbol{a_i^t} \cdot \boldsymbol{h_i}
\tag{2.37}
$$

As for dealing with OOV words, in addition to this attention mechanism augmented with coverage, the authors apply a pointer network [Vinyals et al., 2015], which results in a model that is capable of copying words from the original text and also of generating new words for the summaries. They define the generation probability $p_{gen} \in [0, 1]$ for timestep t as the probability of generating a word from the vocabulary, versus copying a word from the source text:

$$
p_{gen} = \sigma(\boldsymbol{w_{h^*}^\intercal} \cdot \boldsymbol{h_t^*} + \boldsymbol{w_s^\intercal} \cdot \boldsymbol{s_t} + \boldsymbol{w_x^\intercal} \cdot \boldsymbol{x_t} + b_{ptr})
\tag{2.38}
$$

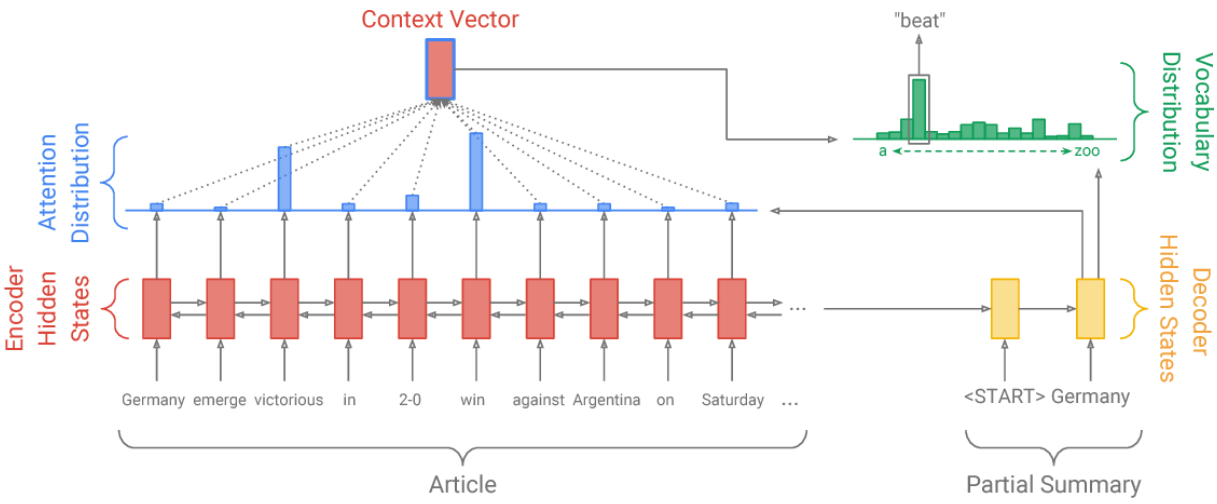


Figure 2.12: Illustration for the pointer-generator network architecture, adapted from See et al. [2017]

.

In the previous expression, $\boldsymbol{w_{h^*}}$ $\boldsymbol{w_s}$, $\boldsymbol{w_x}$ and scalar b_{ptr} are learned parameters. The resulting probability distribution is calculated over the union of the vocabulary and all words appearing in the source document:

$$P(w) = p_{gen} P_{vocab}(w) + (1 - p_{gen}) \sum_{i:w_i=w} \boldsymbol{a_i^t} \tag{2.39}$$

where P_{vocab} is a probability distribution over only the words in the vocabulary, obtained through two linear models fed with the concatenation of the context vector $\boldsymbol{h_t^*}$ and the decoder state $\boldsymbol{s_t}$:

$$P_{vocab} = \text{softmax}(\boldsymbol{W^2}(\boldsymbol{W^1}[\boldsymbol{s_t}, \boldsymbol{h_t^*}] + \boldsymbol{b^1}) + \boldsymbol{b^2}) \tag{2.40}$$

This model is illustrated in Figure 2.12. The loss function used to train the final model targets both the primary loss for the model in general and penalizes repeatedly attending to the same locations, weighted by a hyperparameter λ:

$$\text{loss}_t = -\log P(w_t) + \lambda \sum_i \min(\boldsymbol{a_i^t}, \boldsymbol{c_i^t}). \tag{2.41}$$

The current state-of-the-art in a variety of NLP tasks is represented by applications of the Transformer architecture, previously described in section 2.2.4. Notably, the applicability of pre-trained Transformer-based language models to a variety of downstream tasks has inspired the development of summarization methods that combine approaches from extractive and abstractive summarization.

Liu and Lapata [2019] showcase how BERT can be fine-tuned for both extractive and abstractive summarization tasks. BERT, in its original formulation as masked language model, produces representations that are grounded to tokens instead of sentences, making its application to extractive summarization less straightforward. BERTSum, illustrated in figure 2.13, extends the original BERT model architecture with the following enhancements:

- A [CLS] token is inserted at the start of every sentence in the source document. Each [CLS]

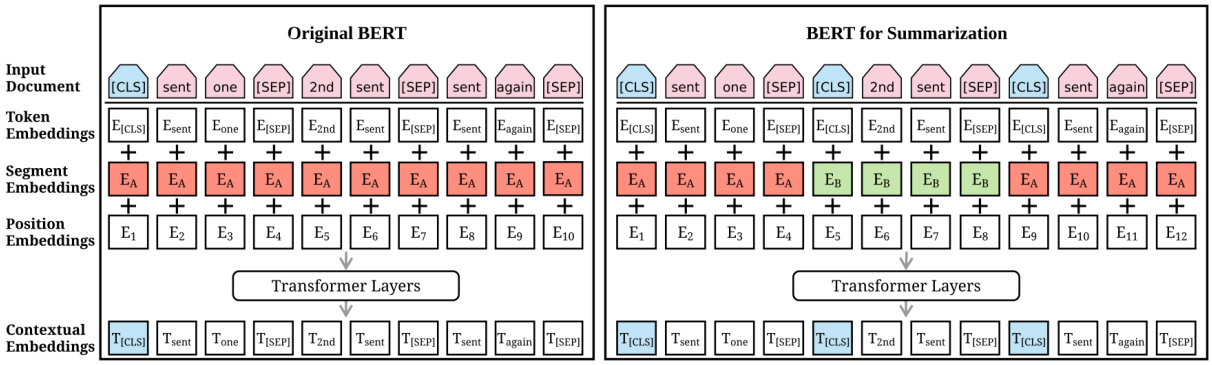


Figure 2.13: Architecture of the original BERT model (left) and BERTSum (right), adapted from Liu and Lapata [2019]

.

token incorporates features for the sentence that precedes it, allowing it to be used as a sentence representation.

- To distinguish multiple sentences within a document, a sentence embedding E_A or E_B is assigned to sentence i depending on whether i is odd or even, respectively. This allows the model to learn document representations hierarchically, where discourse can be progressively generalized along Transformer layers.

For extractive summarization, BERTSum stacks several Transformer layers on top of BERT outputs, generating document-level features for extractive summaries, described by the following equation:

$$\begin{aligned} \tilde{\boldsymbol{h}}^{\boldsymbol{l}} &= \mathrm{LN}(\boldsymbol{h}^{\boldsymbol{l-1}} + \mathrm{MHAtt}(\boldsymbol{h}^{\boldsymbol{l-1}}) \\ \boldsymbol{h}^{\boldsymbol{l}} &= \mathrm{LN}(\tilde{\boldsymbol{h}} + \mathrm{FFN}(\tilde{\boldsymbol{h}}^{\boldsymbol{l}} + \mathrm{FFN}(\tilde{\boldsymbol{h}}^{\boldsymbol{l}})) \end{aligned} \tag{2.42}$$

In the previous expression, $\boldsymbol{h}^{\boldsymbol{0}}$ is the positional embedding associated with a given sentence vector output from BERTSum, and LN, MHAtt and FFN correspond respectively to the linear, multi-headed attention, and feed-forward network layers of the Transformer model. The final output layer is a sigmoid classifier, described by the following expression:

$$\hat{\boldsymbol{y}}_{\boldsymbol{i}} = \sigma(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{h}_{\boldsymbol{i}} + \boldsymbol{b}_{\boldsymbol{o}}) \tag{2.43}$$

In the previous expression, $\boldsymbol{W}_{\boldsymbol{o}}$ is a learned weight matrix, $\boldsymbol{h}_{\boldsymbol{i}}$ is the vector for sentence i generated by the Transformer's top layer, and $\boldsymbol{b}_{\boldsymbol{o}}$ is a bias term. The model is trained in the extractive summarization context by minimizing the cross entropy loss of prediction $\hat{\boldsymbol{y}}_{\boldsymbol{i}}$ against the true label $\boldsymbol{y}_{\boldsymbol{i}}$.

For abstractive summarization, the authors employ a standard encoder-decoder approach, where the encoder is the pre-trained BERTSum and the decoder is a standard Transformer. To avoid a mismatch between the encoder and the decoder (as the encoder is pre-trained while the decoder is trained from scratch), each module features its own gradient-based optimizer. In addition, the same model is fine-tuned in a two-stage approach, where the encoder is fine-tuned on the extractive summarization task and subsequently fine-tuned on the abstractive summarization task. A similar mechanism is employed in this work for hybrid summarization.

Tretyak and Stepanov [2020] experiment further with combinations of extractive and abstractive approaches, applied to the summarization of long scientific texts. The authors' proposed model consists of two components: a BERT-based classifier that selects sentences from the source document which should be included in the summary, and an abstractive language model that conditionally generates a summary from the source text conditioned with the abstractive summary. The experiments for the latter component are carried out on GPT-2 [Radford et al., 2019] and BART [Lewis et al., 2020] models.

Extractive summarization is reduced to a sequence matching task, where each sentence in the ground truth summary is matched with each sentence from the source text. The ROUGE-F1 score for each sentence-pair is calculated, and the two highest scoring sentences are considered as positive examples. Two negative examples are randomly sampled from the remaining sentences. The input for the extractive model is formulated as described in the following expression:

$$\mathrm{input}_{\mathrm{ext}} = [\mathrm{CLS}]; \boldsymbol{s}; [\mathrm{CLS}]; \hat{\boldsymbol{s}} \tag{2.44}$$

In the previous expression, [CLS] is BERT's special classification token, $\boldsymbol{s}$ denotes a sentence from the ground truth summary, $\hat{\boldsymbol{s}}$ denotes the corresponding candidate sentence to be classified as matching the ground truth sentence, and ; denotes the concatenation operator.

The input to the abstractive summarization model is conditioned on the sentences extracted through the previously fine-tuned extractive summarization model. The input sequence to the abstractive model is defined by the following expression:

$$\mathrm{input}_{\mathrm{abs}} = \boldsymbol{t_c}; \boldsymbol{t_s} \times \mathrm{mask}_{\mathrm{segment}} \tag{2.45}$$

In the previous expression, $\boldsymbol{t_c}$ denotes the conditioning text, which is concatenated with the target summary $\boldsymbol{t_s}$. A segment mask $\mathrm{mask}_{\mathrm{segment}}$ is applied on the final sequence, identifying which part of the sequence is conditioning text and which is target text.

Conditioning was made on the following conditioning texts: the extractive summary; the introduction of each scientific document; the introduction concatenated with the conclusion; and the introduction concatenated with both the extractive summary and the conclusion. The authors' intuition is that the introduction and conclusion of a scientific document concentrate the most valuable information to be included in a summary: the introduction often contains a problem statement and details about proposed solutions, while the conclusion usually contains a restatement of the applied methods and a discussion of the results. The authors show that conditioning the abstractive model on the introduction concatenated with both the extractive summary and the conclusion leads to the better results.

2.4.2 Summarization of Online Forum Discussions

Due to the lack of large-scale annotated datasets, the task of online discussions summarization in a supervised learning approach is underrepresented in academic studies. Traditional approaches are commonly based on multi-document summarization strategies, often employing statistical metrics that

measure the representativeness of a sentence relative to the rest of the discussion, such as Integer Linear Programming (ILP) [Gillick and Favre, 2009]. Ding and Jiang [2015] conducted a survey on extracting opinionated summaries from online forum threads. Their study defines a set of features to capture relevance, text quality and subjectivity. They have found that traditional text summarization techniques, which only consider representativeness, very often select low quality sentences.

Bhatia et al. [2014] were the first to frame the task of extracting summaries from online discussions as a summarization problem. Their intuition is that the role of each individual post in a discussion is a crucial feature for a summarization model to consider when extracting information. Thus, they define a summary as a collection of important posts from a discussion, where each post is annotated with a specific *dialog act* that represents the role of the post in the discussion. Features for classification include, for example, the number of unique words in a post, its similarity with the title, and the position of the post in the discussion.

More recently, Tarnpradab et al. [2017] were the first to employ an approach based on deep neural networks for the extractive summarization of online discussions. The authors employ a hierarchical attention network (HAN) [Yang et al., 2016], which is trained to learn sentence representations by attending to important words and subsequently learn representations for the entire discussion in a hierarchical manner. Sentences are encoded by replacing each word with a pre-trained embedding generated through Word2Vec and feeding them to a bidirectional LSTM. The output of each layer of the LSTM is concatenated to produce a sentence representation $\boldsymbol{h_t}$ for the t-th word. In addition, an attention mechanism is employed by introducing a trainable weight vector $\boldsymbol{u_w}$ for each word w. This vector is used to generate a scalar value α_t indicating word importance, described by the following expressions:

$$\boldsymbol{u_t} = \tanh(\boldsymbol{W^a h_t} + \boldsymbol{b^a}) \tag{2.46}$$

$$\alpha_t = \frac{\exp(\boldsymbol{u_t^\intercal u_w})}{\sum_t \exp(\boldsymbol{u_t^\intercal u_w})} \tag{2.47}$$

In the previous expression, the inner product $\boldsymbol{u_t^\intercal u_w}$ models the importance of the t-th word. The final sentence vector $\boldsymbol{s_i}$ is generated by computing a weighted sum of words representations, defined as follows:

$$\boldsymbol{s_i} = \sum_t \alpha_t \boldsymbol{h_t} \tag{2.48}$$

A document is a sequence $\boldsymbol{s} = [\boldsymbol{s_i}, \ldots, \boldsymbol{s_n}]$ of sentences encoded through equation 2.48 and of length n. This sequence is equally fed through a bidirectional LSTM and an attention layer, generating a representation for an entire discussion in the same way that sentence representations are generated. Finally, each sentence in a document is represented by concatenating the corresponding sentence and document representations.

Building upon the aforementioned study, Magooda and Marcjan [2020] enhance the extractive model by applying additional attention vectors that give special weight to the sentences in the beginning of a discussion. The intuition is that the first post of a discussion concentrates topical information that is

crucial for contextualizing the model. Each sentence in the first post is paired with a sentence from the rest of the discussion. A similarity matrix $\boldsymbol{S} \in \mathbb{R}^{m \times n}$ is computed for each sentence pair $\boldsymbol{s_{ij}} = [\boldsymbol{h_i^d}; \boldsymbol{h_j^b}; \boldsymbol{h_i^d} \otimes \boldsymbol{h_j^{ip}}]$. In the aforementioned expressions, n is the number of sentences in the first post, m is the total number of sentences in the document, ; is the concatenation operator, $\boldsymbol{h_i^d}$ is the representation of the i-th sentence in the document, and $\boldsymbol{h_j^b}$ is the representation of the j-th sentence in the first post. Each row and column of $\boldsymbol{S}$ is normalized through a softmax function, producing two new matrices $\boldsymbol{S_1}$ and $\boldsymbol{S_2}$. Bidirectional attention is then calculated as $\boldsymbol{A} = \boldsymbol{S_1} \cdot \boldsymbol{h^b}$ and $\boldsymbol{B} = \boldsymbol{S_1} \cdot \boldsymbol{S_2^\intercal} \cdot \boldsymbol{h^b}$, where $\boldsymbol{A}$ and $\boldsymbol{B}$ represent document-to-first-post attention and first-post-to-document attention respectively. The final sentence representation for sentence i is computed by:

$$\boldsymbol{g_i^d} = [\boldsymbol{h_i^d}; \boldsymbol{A_i}; \boldsymbol{h_i^d} \otimes \boldsymbol{A_i}; \boldsymbol{h_i^d} \otimes \boldsymbol{B_i}] \tag{2.49}$$

This mechanism is integrated with SummaRuNNer [Nallapati et al., 2017] and SiATL [Chronopoulou et al., 2019]. SummaRuNNer is an auto-regressive model, meaning that the model's previous decisions affect subsequent decisions. SiATL, by contrast, is non-auto-regressive, so each decision is independent of each other. The authors report better results on SiATL.

2.4.3 Summarization of Long Scientific Documents

The task of summarizing long scientific texts using deep neural models was first explored by Cohan et al. [2018], who employed an abstractive discourse-aware model based on an encoder-decoder architecture. This model takes advantage of the general structure of a scientific article, attending to different discourse sections. The encoder is a hierarchical RNN that learns representations for each discourse section and subsequently for the entire document. The decoder is enhanced with special attention vectors that attend to the relevant discourse section. To address the problem of unknown token prediction, the model includes an additional binary classifier that decides whether the next word should be generated from the vocabulary or copied from the source document. Xiao and Carenini [2019] expand this approach to extractive summarization by combining local (section-level) and global (document-level) context. At the document-level, a bidirectional GRU produces sentence representations by concatenating the backward forward hidden states for each sentence. Document representations act as global context, computed by concatenating the final state of the forward and backward GRU. Local context is captured through the LSTM-minus method, where each section is represented as the subtraction between the hidden states of the start and the end of that section. Finally, Tretyak and Stepanov [2020] combine extractive and abstractive approaches. The authors' proposed model consists of two components: a BERT-based classifier that selects sentences from the source document which should be included in the summary, and an abstractive language model that conditionally generates a summary from the source text conditioned with the abstractive summary. The experiments for the latter component are carried out on GPT-2 [Radford et al., 2019] and BART [Lewis et al., 2020] models.

2.5 Overview

This chapter presents the theoretical foundations of machine learning algorithms with neural networks (Section 2.1) and a history of strategies for learning text representations for such algorithms (Section 2.2). We describe common metrics for the evaluation of summarization models in section 2.3. Finally, this chapter described methods proposed in the literature for extractive and abstractive text summarization models powered by neural networks (Section 2.4.1), and presented relevant studies aimed at applying these methods to the domains of online discussions (Section 2.4.2) and long scientific documents (Section 2.4.3).

Chapter 3

Proposed Approach

This work carries out a series of comparative experiments related to the tasks of extractive and abstractive text summarization using Transformer models. In addition, we advance a novel hybrid summarization approach based on a single T5 model. This approach is mainly evaluated in the tasks of summarizing online discussions. To validate the transferability of our approach, we carry out identical experiments in the domain of long scientific documents. The following sections describe each experiment and steps carried out in each one. We start with an introduction to how text is pre-processed and turned into input samples for each model. We then describe each of our approaches to extractive and abstractive summarization. We conclude with a description of how these approaches are combined to build our hybrid framework for text summarization.

3.1 Pre-Processing Unstructured Text for Transformer models

Before the emergence of contextual embedding models, it was common to apply noise removal and normalizing steps to data before generating representations, by removing stop-words, which are words that are very common but carry no meaning by themselves. For example, *the*, *is*, *at*, *which*, and *on* are commonly considered stop-words in studies dealing with the English language. For representations that depend on the surrounding context of a word, it is no longer beneficial to remove stop-words, as doing so often removes context from the text that is crucial for the generation of accurate contextual representations. Datasets for extractive and abstractive summarization, although containing the same data, are considered separately.

At the heart of our experiments is the Transformers library [Wolf et al., 2019] by Hugging Face. Transformers provides ready-to-use abstractions for state-of-the-art Transformer based models through its API, as well as pre-trained tokenizer classes that split a string into lowercase tokens and subsequently convert these tokens into vocabulary indices, which are used as inputs for the models. Transformers provides one tokenizer for each model. These tokenizers are based on WordPiece [Wu et al., 2016]. WordPiece is a data-driven pre-trained tokenizer that deterministically splits a word sequence into a sequence of sub-word units. This allows a model to consider arbitrary OOV words, as the final sub-

words are always in the vocabulary. The final sequence can be used to recover the original sequence. An example of a word sequence and the corresponding WordPiece sequence is described as follows:

- **Original sequence:** Lorem ipsum is used to test text.
- **WordPiece sequence:** lore ##m ip ##sum is used to test text .

In the above example, the words "Lorem" is broken in into two word pieces "lore" and "##m", and the word "ipsum" is broken into two word pieces "ip" and "##sum". The other words remain as single word pieces. "##" is added to mark word pieces that belong to the same word as the previous word piece in the sequence, allowing the original sequence to be recovered.

In particular, the tokenizer used for T5 is based on SentencePiece [Kudo and Richardson, 2018], which uses a different tokenization scheme:

- **Original sequence:** Lorem ipsum is used to test text.
- **SentencePiece sequence:** _lor e m _ ips um _is _used _to _test _text .

In this case, "_" is added to mark whitespace, which is also handled as a basic token explicitly, and the beginning of a word. The tokenizers also take care of batching and padding samples together for training, as well as adding model-specific special tokens.

3.2 Extractive Summarization

We reduce the task of extractive summarization to a sentence classification task. Let $\boldsymbol{d} = [\boldsymbol{s_1}, \ldots, \boldsymbol{s_n}]$ be the source document defined as a sequence of n sentences, and $\boldsymbol{y} = [y_1, \ldots, y_n]$ be a sequence of corresponding binary labels, where 1 indicates the sentence is in the summary and 0 otherwise. The goal of extractive summarization is to find the most probable sequence of labels given the sentences of a source document:

$$\underset{\boldsymbol{y} \in \mathcal{Y}}{\operatorname{argmax}}\, p(\boldsymbol{y}|\boldsymbol{d}) \tag{3.1}$$

In the previous expression, $\mathcal{Y}$ is the set of all possible label sequences. Each labeling decision is independent, and as such $p(\boldsymbol{y}|\boldsymbol{d}) = \prod_{i=1}^{n} p(y_i|\boldsymbol{d})$. This is called a non-auto-regressive model, contrasting with an auto-regressive model which conditions future decisions on past decisions.

3.2.1 DistilBERT baseline

Before starting our experiments leveraging T5's transfer learning capabilities, we set up a baseline using a simpler model for comparison. We leveraged a pre-trained DistilBERT [Sanh et al., 2019] model extended with an additional linear layer for binary classification. DistilBERT is a faster and lighter version of BERT, featuring less parameters while retaining language understanding capabilites. We chose this model after some trial and error along with ALBERT [Lan et al., 2019] and BERT, after which DistilBERT

showed the best results despite its comparatively smaller size. Moreover, the larger versions of BERT and ALBERT were too computationally demanding for our setup, making repeated experimentation prohibitive. Note that due to timing constraints, we did not apply this baseline to the arXiv dataset.

An input sample, corresponding to sentence $\boldsymbol{s_i}$ is formulated as follows:

$$\text{input}_{\text{BERT}} = [\text{CLS}]; s_i; [\text{SEP}] \tag{3.2}$$

In the previous expression, [CLS] and [SEP] are BERT-specific tokens, and ; is the concatenation operator. DistilBERT outputs a sequence of token representations $\boldsymbol{R_i} \in \mathbb{R}^m \times \mathbb{R}^n$, where m is the number of word pieces in s_i and n is the length of the token representation. For DistilBERT, $n = 768$. We take the representation of the [CLS] token, $\boldsymbol{R_{i_0}}$, for its adequacy to classification tasks, and feed it to a linear layer. The final output is fed to a softmax function to obtain probabilities for binary classification. This approach is described in the following expression:

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{W}\boldsymbol{R_{i_0}} + \boldsymbol{b}) \tag{3.3}$$

In the previous equation, $\boldsymbol{W} \in \mathbb{R}^n \times \mathbb{R}^2$ is the final weight matrix for the classifier and $\boldsymbol{b}$ is a bias term. The vector $\hat{\boldsymbol{y}}$ contains the probabilities of the positive and negative classes. To produce an extractive summary, we run every sentence in document $\boldsymbol{d}$ through the model, and rank them by the probability of the positive class. Finally, we keep only the top ranked sentences, setting a threshold equal to the average number of sentences in the reference summaries.

3.2.2 Extractive Approach with T5 by Leveraging Target Words

For T5, an input sample for extractive summarization corresponding to a sentence $\boldsymbol{s_i}$ is formulated as follows:

$$\texttt{important sentence: } s_i \texttt{ </s>} \tag{3.4}$$

In the previous expression, we add the prefix ”important sentence:” to the input sample to provide task-specific context to the model. </s> corresponds to the end-of-sentence token.

As mentioned in section 2.2.4, T5 is a purely text-to-text language model. As such, it does not output numerical values that can be directly transformed to classification probabilities. Nevertheless, the original authors have fine-tuned T5 on a number of classification tasks, simply by targeting specific words with positive and negative connotation, and then asking the model to generate those words. We initially followed this reasoning by replacing the 1 and 0 labels with the words ”true” and ”false”, targeting these words while training. However, directly asking the model to generate these words on inference led to very poor results (i.e., the vast majority of the generated words were ”false”, producing no summaries). We redefined our approach by following Nogueira et al. [2020]: instead of targeting ”true” and ”false” directly, we extract the probabilities given by T5’s language model and train and perform inference based on the relative probabilities of generating either of the words.

Figure 3.1: Illustration of our abstractive summarization model with an extractive step. We use our best extractive model to reduce the training set to their extractive summaries. We then train a distinct abstractive model using the extractive summaries as input.

Given an input sequence $\boldsymbol{s} \in \mathcal{V}^l$ of tokens that are part of vocabulary $\mathcal{V}$ and of length l, the T5 model generates a sequence $\boldsymbol{o}$ of tokens of user-defined length m, i.e., $\boldsymbol{o} \in \mathcal{V}^m$. The model's output is a matrix $\boldsymbol{R} \in \mathcal{V}^m \times \mathcal{V}^n$, where n is the size of $\mathcal{V}$ and every entry $\boldsymbol{R}_{ij}$ is the logit (a "prediction score") that the i-th output token is equal to the j-th vocabulary entry. In our approach, we evaluate the pair of logit values $t = \boldsymbol{R}_{0,'\text{true}'}; f = \boldsymbol{R}_{0,'\text{false}'}$. We then compute the activation $\hat{y} = \sigma(t - f)$, where σ denotes the sigmoid function, interpreting this value as the probability of the positive class. Summaries are then produced in the same way as described in section 3.2.1.

3.3 Abstractive Summarization

Abstractive summarization is made possible by T5's conditional generation capabilities. Let $\mathcal{X}$ be the set of all possible input sequences and $\mathcal{Y}_n$ be all possible sequences of user-defined length n. The task of abstractive summarization finds the optimal sequence $\boldsymbol{y} \in \mathcal{Y}_n$ under a scoring function $s : \mathcal{X} \times \mathcal{Y}_n \mapsto \mathbb{R}$:

$$\underset{y \in \mathcal{Y}_N}{\operatorname{argmax}}\, s(\boldsymbol{x}, \boldsymbol{y}) \tag{3.5}$$

In the previous expression, $\boldsymbol{x} \in \mathcal{X}$. We carry out two experiments for abstractive summarization:

- We take the source document as-is and truncate it to the first 512 word pieces.
- Following Subramanian et al. [2019] and Tretyak and Stepanov [2020], and as illustrated in figure 3.1 we perform an extractive step before training, using another T5 model trained under the extractive context, to reduce the source document to its important sentences. The model takes as input the first 512 word pieces of this extractive summary.

Our input for this task is the following:

$$\texttt{summarize: } d \texttt{ </s>} \tag{3.6}$$

where d is the document to summarize and `summarize:` is the pre-trained prefix for the abstractive summarization task. We train the abstractive model on the reference summaries for each dataset. Training is performed in "teacher-forcing" fashion [Williams and Zipser, 1989], where the input and target sequences are directly fed to the model.

We use beam search decoding to generate summaries, configured with 10 beams. Decoding stops once all beams generate the end of sequence token or reach a maximum sequence length, which is

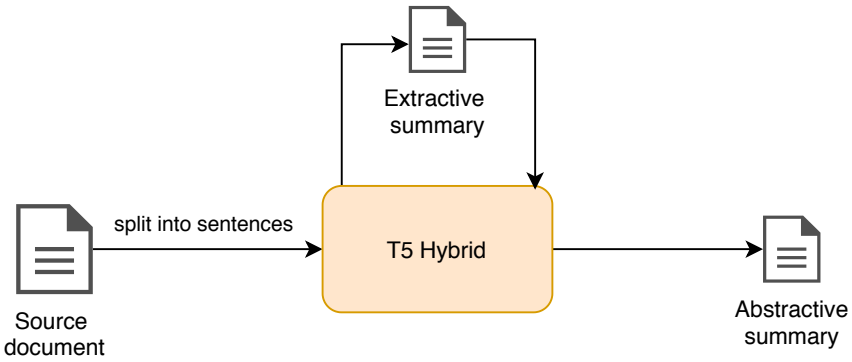


Figure 3.2: Illustration for our iterative hybrid approach for summarization. Source documents are first reduced to an extractive summary. Subsequently, the same model produces an abstractive summary from the selected sentences.

set as equal to the average number of words in the reference summaries. We arrived at this approach after some trial and error, where we experimented with greedy decoding and less beams, which yielded slightly worse results.

3.4 Combining Extractive and Abstractive Summarization

We advance a novel hybrid framework for summarization which iteratively learns to extract important sentences from a document and subsequently generate an abstractive summary from the extractive summary, by combining the approaches described earlier in this section. We leverage T5's unified text-to-text interface to train the same model in both tasks. Our framework is illustrated in figure 3.2 and described as follows:

1. We perform one training epoch in the extractive context.
2. Using the model trained in the previous step, we reduce the training set to a set of extractive summaries.
3. We perform one epoch in the abstractive context, taking as input the extractive summaries generated in the second step.

This process is repeated for the defined number of epochs. The resulting model can be used in both extractive and abstractive summarization tasks, and is evaluated in both these contexts. Note that, to evaluate this model in the abstractive context, it was necessary to instruct the decoder to avoid generating the ”true” and ”false” tokens.

3.5 Training Setup

All models were trained under automatic mixed precision to reduce memory requirements. We followed the recommendations by Mosbach et al. [2020] for fine-tuning BERT, from the intuition that they would also apply to the training of other Transformer-based language models. Specifically, we use a batch

size of 16, and employ the AdamW optimizer [Loshchilov and Hutter, 2017] for training. Sequences are padded, with special padding tokens, to the length of the longest sequence in the batch, or to the maximum length supported by the model. Regarding the choice of learning rate, we setup a schedule that increases the learning rate linearly from zero to $2\mathrm{e}{-5}$ for the first 10% of steps and linearly decreases it back to zero afterwards. We set the weight decay value to 0.01 and clip gradient norms to 1.0 to avoid exploding gradients. We train for 20 epochs and report results on the epoch that yields the higher ROUGE scores on the validation datasets. Note that, due to timing constraints, for the arXiv dataset, we train for only one epoch as the dataset is too large to be trained in a convenient time-frame. In addition, we were unable to fit a batch of 16 documents in the abstractive contexts for the arXiv dataset, and as such we used a batch size of 8 and accumulated gradients each 2 steps to simulate a batch of 16 samples. Finally, for the hybrid context, we trained each of the two modules with a distinct AdamW optimizer, with identical hyperparameters. The loss used in all experiments was the cross-entropy loss.

3.6 Overview

This chapter presented the extractive and abstractive approaches we considered for our comparative study in text summarization, as well as our novel hybrid approach that combines the two strategies. Furthermore, it describes specific implementation details for our training procedures.

Chapter 4

Experimental Evaluation

This chapter describes the evaluation procedures we carry out to assess the performance of our models. We describe the datasets used to evaluate our approaches in section 4.1, as well as a description of the steps taken to convert these datasets into a format that can be easily used in our experiments. We present the results yielded by each of our experiments in section 4.2.

4.1 Datasets and Evaluation Methodology

This section lays out a simple statistical characterisation of both datasets used in this study, followed by a description of the steps taken to convert the original data into a format that could be easily loaded into our experimental setup. Finally, we lay out the evaluation metrics we used to measure the adequacy of the summaries generated in all our experiments. A simple statistical analysis of these datasets is presented in table 4.1. These values have been obtained by manual inspection. For evaluating performance in the context of summarizing online discussions, we applied our approaches to a dataset of discussions taken from the TripAdvisor user forum [Bhatia et al., 2014, Tarnpradab et al., 2017], described in section 4.1.1. To evaluate the transferability of our methods to other domains, we carry out the same experiments on a dataset of long scientific documents taken from the scientific repository arXiv [Cohan et al., 2018], described in section 4.1.2.

Both these datasets feature textual documents paired with human-generated references. While evaluating text similarity is well understood, for example using the metrics described in section 2.3, evaluating the actual reference summary is challenging. Trained summarization methods have to rely on the existence of some reference summary and the trained model should approximate this reference summary as it trains. However, real world datasets might not be ideal and suffer from multiple issues. Firstly, the TripAdvisor dataset contains two different summaries for each discussion. This would indicate that even a human that generates this reference summary has no one-true solution to what the *best* summary is and training a model that can provided two very different outputs for the same input is challenging. Furthermore, the dataset also includes several typos and linguistic problems. For example, in one sample (see Appendix A), the author of the reference summary writes *in **x** modern and inviting restaurant*, which

Dataset	Vocab. size	Sentences /doc	Words /doc	Words /sentence	Sentences /summary	Words /summary	Words /sentence (summary)
TripAdvisor	23414	56	974	16	11	217	18
arXiv	200000	207	4938	29	9	220	34

Table 4.1: Metrics for the TripAdvisor and arXiv datasets. Note that the all entries, except for documents and vocabulary size, are average values.

should read *in **a** modern and inviting restaurant*. Finally, the quality of the reference summaries might also vary significantly between different summaries, even if they were generated by the same human. While some of these problems might be detectable, e.g., typos, verifying entire datasets that might be hundreds of thousands of entries in size is not feasible in practice. Furthermore, evaluating the quality of a reference summary requires time and a human judgement by the individual researcher, which would not be replicable across varying authors. Moreover, remaining consistent across large sets of text, even if a single person were to summarize all 215 thousand articles in the arXiv dataset, is not likely without at least some linguistic inconsistency.

Accordingly, training networks to closely match inherently flawed data will, most likely, also yield flawed summaries that require large parameter spaces to handle the human variance in the summaries. Consequently, it would be important for future research to not rely as much on human generated reference data and instead move towards unsupervised learning, at least to some extent.

4.1.1 TripAdvisor

The main dataset used for this study is from the domain of online discussions. It consists of 700 discussions from the New York City section of TripAdvisor's user-managed online forum[1], first introduced by Bhatia et al. [2014] and expanded by Tarnpradab et al. [2017]. Of these, following Magooda and Marcjan [2020], 100 discussions belong to the test set, while 100 discussions from the remainder are randomly sampled to build the validation set. The remaining 500 discussions correspond to the training set. The original authors parsed each document as sentences using Stanford Core NLP [Manning et al., 2014]. A discussion features a title (structured as a list of sentences) and the discussion body (structured as a list of posts, each of which is a list of sentences), and is paired with three summaries: one extractive and two abstractive. The extractive summary is simply a list of indices, corresponding to the positions of the sentences in the source document which belong to the extractive summary. Note that none of the summarization approaches consider the title of the discussion for the generation of summaries. Thus the title is not included in the inputs to the models. The concatenation of all sentences in the discussion is considered as the source document for the summarization task.

We iterate through this parsed data and write it to an easily-traversable JSON format, which can be loaded directly onto a Python dictionary. The dictionary format for the samples in this dataset is described in listing 4.1. The entire dataset is small enough to fit in available memory, and as such can be directly batched, pre-processed and fed to the models. The list of indices corresponding to the

[1] `http://www.tripadvisor.com/ShowForum-g28953-i4-New_York.html`

extractive summary is converted into a list of zeros and ones.

```
{
 "title": list[string],
 "posts": list[list[string]],
 "extractive_indices": list[int],
 "human_summaries": list[string], #len(human_summaries) = 2
}
```

Listing 4.1: Format of a sample from the TripAdvisor dataset.

4.1.2 arXiv

In order to validate the transferability of our approaches, we perform identical experiments on a dataset from the domain of long scientific documents. This dataset was first compiled by Cohan et al. [2018] with the goal of generating the abstract given the document text. It consists of 214294 documents from the electronic scientific repository arXiv[2]. Of these, 201427 documents belong to the training set, while the validation and testing sets comprise 6431 and 6436 documents, respectively.

The dataset preserves the discourse structure of each scientific article, by splitting the text into its original sections; however, we consider the entirety of the document body as an input sample and pay no special attention to discourse structure. Each sample contains the arXiv unique identifier for the article, the sentence-split text of the abstract and the sentence-split text of the document body, in addition to discourse structure information. The authors converted the original LaTeX files to plain text, removing figures and tables and normalizing mathematical formulas and citation markers with special tokens. All sections after the conclusion were removed. Each document is in its own file, and we load each file into our experiments while training.

For extractive summarization, we leverage the work of Xiao and Carenini [2019], who have assigned binary labels to each sentence in each document. We iterate through each document and extract all sentence-label pairs into a separate file. To avoid loading the entire file into memory, we construct a special dictionary which maps the index of each line in the file to the byte-offset of that line. This also allows us to load samples randomly instead of sequentially processing all sentences in a fixed order, which would be otherwise necessary to process this dataset. To access a specific sample, we use the dictionary to obtain the corresponding byte-offset and seek to that point in the file.

4.1.3 Evaluation Methodology

Evaluation of our summarization models is performed by comparison with corresponding human references. We evaluate all methods using two of the standard evaluation metrics described in section 2.3. We report average scores for both ROUGE and BERTScore. Despite the demonstrated inadequacy of

[2] http://arxiv.org/

Method	Params.	Type	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore
Tarnpradab et al. [2017]	-	Ext.	37.60	14.40	33.80	–
Magooda and Marcjan [2020]	-	Ext.	46.50	28.53	44.65	–
DistilBERT	66M	Ext.	38.96	15.85	24.77	39.11
T5 Classifier	60M	Ext.	35.87	13.54	22.56	37.02
T5 Hybrid	60M	Ext.	36.38	14.14	23.21	37.20
T5 Summarizer	60M	Abs.	35.47	12.65	21.10	38.16
T5 After Extractive	120M	Abs.	37.05	14.38	21.72	39.15
T5 Hybrid	60M	Abs.	36.01	13.32	21.21	38.72

Table 4.2: Scores for the TripAdvisor dataset. Our contributions are in bold.

Method	Params.	Type	ROUGE-1	ROUGE-2	ROUGE-L	BERTScore
Xiao and Carenini [2019]	-	Ext.	43.62	17.36	29.14	–
Tretyak and Stepanov [2020]	110M	Ext.	45.40	20.90	33.70	–
T5 Classifier	60M	Ext.	40.53	12.66	22.13	43.14
T5 Hybrid	60M	Ext.	40.41	12.24	21.62	42.92
Cohan et al. [2018]	-	Abs.	35.80	11.05	31.80	–
Tretyak and Stepanov [2020]	249M	Abs.	45.30	25.10	36.20	–
T5 Summarizer	60M	Abs.	31.50	8.47	19.82	31.27
T5 After Extractive	120M	Abs.	38.52	13.23	23.84	39.61
T5 Hybrid	60M	Abs.	34.15	10.47	22.31	35.72

Table 4.3: Scores for the arXiv dataset. Our contibutions are in bold.

ROUGE for evaluating summarization models, we report results in such metric as a means of comparison with previous works. BERTScore is reported for comparison between our approaches. We report the rescaled score, as described in section 2.10. For calculating ROUGE scores, we leverage the py-rouge[3] library, while for BERTScore we use the official implementation[4]. We generate one summary for each document in the dataset, and compare it with its respective human-generated reference summary.

4.2 Experimental Results

This section presents a comparison of the results yielded by each of our experiments, under the metrics described previously. By analysing these results, we attempt to answer the research questions: RQ-I) *Which approach leads to better results in a summarization task?* and RQ-II) *Is this approach applicable to other domains?*. RQ-I) is answered by comparing metric scores between the results of our experiments on the TripAdvisor dataset. RQ-II) is answered by applying these steps to the arXiv dataset.

As mentioned in section 1.3, the standard metrics used for evaluating the results of our experiments were ROUGE and BERTScore. Better performing models generate summaries that more accurately represent the information that is laid out in the reference summaries. For ROUGE, this means that generated summaries which have a higher amount of overlap with the reference summaries are rewarded. As BERTScore considers a soft similarity measure instead of n-gram overlap, for this metric high performance can be achieved with paraphrases of the same information.

[3] https://github.com/Diego999/py-rouge
[4] https://github.com/Tiiiger/bert_score

The average ROUGE and BERTScore values for TripAdvisor and arXiv are displayed in table 4.2 and table 4.3, respectively. We include results from previous studies in summarization tasks performed on the same datasets for comparison. Appendix A lists some examples of generated summaries from the TripAdvisor dataset, for visual inspection.

As expected, the best results in terms of ROUGE scores are consistently achieved by extractive methods; as described previously, extractive summarization merely copies sentences from the source document, and as such always generate summaries with proper grammatical and lexical fluency. However, visually inspecting the extracted summaries for the TripAdvisor dataset reveals their content is often uncontextualized from a discourse perspective, as the extracted sentences are taken from different posts written by different authors, and simply concatenated together to form the final summary. This effect could be mitigated by identifying the posts the extracted sentence came from. Note that this would merely make the extracted summaries more readable, and would not affect the automatic scores.

Abstractive summarization after a pre-processing step using an extractive model yields the best results for both datasets. The main cause of this improvement is, perhaps, the reduction in unimportant information that the abstractive model has to consider when training.

However, our hybrid method does not outperform any of the standalone approaches for summarization. Moreover, evaluation of the hybrid method in an abstractive context shows worse results than abstractive summarization after an extractive step. This is more pronounced in the arXiv dataset, possibly due to this dataset being several orders of magnitude larger than the TripAdvisor dataset, in terms of size, document length, lexical complexity and grammatical variability. We describe a few hypothesis as to why the hybrid model performs worse:

- The version of T5 used in our experiments features a notoriously small amount of parameters, limiting its ability to model more complex discourse. Using larger models, with more parameters, consistently leads to better results in previous works. The judgement of what makes a good summary is inconsistent as the task of summarization is highly subjective, making larger models better at modeling this high variability.
- As the extractive summarization task works on the sentence level, it works on a much larger amounts of samples than the abstractive summarization task. As such, the hybrid model is susceptible to an imbalance while iteratively training on both tasks, potentially undermining the performance under the abstractive context.
- In the case of the arXiv dataset, as each summary is written by the author of the source document, there is much higher variability in the relationships between the summary and the document body, making supervised training significantly harder.

We note that performance of the hybrid model on extractive tasks remains similar to results on standalone extractive models. Notably, none of our approaches are able to match current state-of-the-art.

Chapter 5

Conclusions and Future Work

5.1 Contributions

The main contribution of this work was a set of experimental approaches to automatic text summarization of online discussions, powered by a sequence-to-sequence pre-trained Transformer model, studying strategies for extractive and abstractive summarization, and combinations of both. Each approach was evaluated using standard performance metrics for the evaluation of automatic summarization models. We advanced a novel framework for hybrid summarization that iteratively learns to extract sentences from a document and to generate a summary from such sentences. Furthermore, we applied these approaches to the task of summarizing long scientific documents, to assess the transferability of our approaches. We showed that using two distinct models trained in separate for extractive and abstractive summarization yields the best results in our experiments.

Surprisingly, training a single model iteratively in both tasks does not surpass the remaining approaches in terms of metric scores. The main limitation of this approach, as such, is that it does not represent a worthwhile alternative to more memory-demanding state-of-the-art models, as it does not match them in terms of performance. It is unclear if the poor results demonstrated were due to using a smaller version of the model, which is less expressive due to having fewer parameters, or because the training procedure was inadequate; indicating that fine-tuning a single model in two separate tasks undermines its performance in one of them. One hypothesis for explaining this disparity in performance is that since there are so many more samples to consider when training under an extractive context than under an abstractive one, the model becomes significantly biased towards the extractive task, which would explain the similar performance to the baselines in standalone extractive tasks, but worse in abstractive ones. This work would benefit from ablation studies, in order to isolate the cause of the worse performance of the hybrid model.

Another limitation is the weak validation for transferability of our approaches to other domains. Our experiments with the arXiv dataset were severely undermined by constraints in available computing resources and training time. Therefore, we were not able to perform a full training procedure on the arXiv dataset, as under the available resources a full training procedure had an estimated duration in

the order of weeks. We suspect that fine-tuning our models on this dataset for more epochs and larger batch sizes will yield significantly better results.

5.2 Future Work

In the future, we aim to explore some simple enhancements to our training routine to improve the effectiveness of the hybrid model, such as paying attention to certain parts of each document (e.g. the beginning of an online discussion [Magooda and Marcjan, 2020] or the concluding statements of a scientific paper), or experimenting with transformations of the data (e.g. instead of reducing a document to its most important sentences, simply condition the model with the extractive summary by concatenating it with the source document [Subramanian et al., 2019, Tretyak and Stepanov, 2020]).

Despite failing to produce results matching state-of-the-art, manually inspecting the best summaries generated by our approaches reveal proper fluency and information representation. Therefore, this work would benefit from an additional study comparing the automatic scores yielded by standard metrics with human judgements.

Finally, we would like to refine our current approach to larger datasets, using increased computational resources, that might allow our method to handle data-rich environments better as our constraints to small datasets and models, from a conceptual standpoint, are only based on available resources. In particular:

- Use larger versions of the T5 model, with more parameters, which we speculate will lead to better results on larger, more variable datasets;
- For the arXiv dataset, combine the previous point within a longer training regime, leveraging the validation set; potentially combine this approach with larger batch sizes and single-precision training instead of mixed precision.
- Experiment with other state-of-the-art Transformer-based models with conditional generation capabilities, such as BART [Lewis et al., 2020] and PEGASUS [Zhang et al., 2019].
- Explore applications of our method combined with models designed to work with longer input sequences, such as Longformer [Beltagy et al., 2020], for the summarization of longer documents such as the ones in the arXiv dataset.

Bibliography

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document Transformer. *arXiv preprint arXiv:2004.05150*, 2020.

Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, 2003.

Y. Bengio, H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain. Neural probabilistic language models. In *Innovations in Machine Learning: Theory and Applications*. Springer Berlin Heidelberg, 2006.

S. Bhatia, P. Biyani, and P. Mitra. Summarizing online forum discussions – can dialog acts of individual messages help? In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014.

T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillère, J. Klein, and Y. L. Traon. Got issues? who cares about it? a large scale investigation of issue trackers from GitHub. In *Proceedings of the IEEE International Symposium on Software Reliability Engineering*, 2013.

K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

A. Chronopoulou, C. Baziotis, and A. Potamianos. An embarrassingly simple approach for transfer learning from pretrained language models. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

A. Cohan and N. Goharian. Revisiting summarization evaluation for scientific articles. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation*, 2016.

A. Cohan, F. Dernoncourt, D. S. Kim, T. Bui, S. Kim, W. Chang, and N. Goharian. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2018.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional Transformers for language understanding. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2019.

Y. Ding and J. Jiang. Towards opinion summarization from online forums. In *Proceedings of the International Conference Recent Advances in Natural Language Processing*, 2015.

J. L. Elman. Finding structure in time. *Cognitive Science*, 1990.

A. R. Fabbri, W. Kryściński, B. McCann, C. Xiong, R. Socher, and D. Radev. SummEval: Re-evaluating summarization evaluation. *arXiv preprint arXiv:2007.12626*, 2020.

D. Gillick and B. Favre. A scalable global model for summarization. In *Proceedings of the Workshop on Integer Linear Programming for Natural Language Processing*, 2009.

Y. Goldberg. *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers, 2017.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

A. Ivakhnenko and V. Lapa. *Cybernetic Predicting Devices*. CCM Information Corporation, 1973.

K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 1972.

J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 1952.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

W. Kryscinski, N. S. Keskar, B. McCann, C. Xiong, and R. Socher. Neural text summarization: A critical evaluation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, 2019.

T. Kudo and J. Richardson. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018.

Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2020.

C.-Y. Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, 2004.

P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer. Generating Wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

Y. Liu and M. Lapata. Text summarization with pretrained encoders. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, 2019.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

A. Magooda and C. Marcjan. Attend to the beginning: A study on using bidirectional attention for extractive summarization. *arXiv preprint arXiv:2002.03405*, 2020.

C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics System Demonstrations*, 2014.

W. McKinney. Data structures for statistical computing in Python. In *Proceedings of the Python in Science Conference*, 2010.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

M. Mosbach, M. Andriushchenko, and D. Klakow. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv:2006.04884*, 2020.

R. Nallapati, B. Zhou, C. dos Santos, Ç. Gulçehre, and B. Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of The SIGNLL Conference on Computational Natural Language Learning*, Aug. 2016a.

R. Nallapati, B. Zhou, and M. Ma. Classify or select: Neural architectures for extractive document summarization. *arXiv preprint arXiv:1611.04244*, 2016b.

R. Nallapati, F. Zhai, and B. Zhou. SummaRuNNer: A recurrent neural network based sequence model for extractive summarization of documents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2017.

A. Nenkova and R. Passonneau. Evaluating content selection in summarization: The pyramid method. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, 2004.

J.-P. Ng and V. Abrecht. Better summarization evaluation with word embeddings for ROUGE. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2015.

R. Nogueira, Z. Jiang, and J. Lin. Document ranking with a pretrained sequence-to-sequence model. *arXiv preprint arXiv:2003.06713*, 2020.

C. Olah. Neural networks, types, and functional programming, Sept. 2015a. URL `http://colah.github.io/posts/2015-09-NN-Types-FP/`. (accessed: 11.10.2020).

C. Olah. Understanding LSTM networks, Aug. 2015b. URL `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. (accessed: 11.10.2020).

R. J. Passonneau, E. Chen, W. Guo, and D. Perin. Automated pyramid scoring of summaries using distributional semantics. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2013.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 2019.

M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*, 2018.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 2019.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text Transformer. *Journal of Machine Learning Research*, 2020.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 1958.

A. M. Rush, S. Chopra, and J. Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2015.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

A. See, P. J. Liu, and C. D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, Vancouver, Canada, July 2017.

N. S. Sohoni, C. R. Aberger, M. Leszczynski, J. Zhang, and C. Ré. Low-memory neural network training: A technical report. *arXiv preprint arXiv:1904.10631*, 2019.

S. Subramanian, R. Li, J. Pilault, and C. Pal. On extractive and abstractive neural document summarization with Transformer language models. *arXiv preprint arXiv:1909.03186*, 2019.

S. Tarnpradab, F. Liu, and K. Hua. Toward extractive summarization of online forum discussions via hierarchical attention networks. In *Proceedings of the International Florida Artificial Intelligence Research Society Conference*, 2017.

V. Tretyak and D. Stepanov. Combination of abstractive and extractive approaches for summarization of long scientific texts. *arXiv preprint arXiv:2006.05354*, 2020.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. *Advances in Neural Information Processing Systems*, 2015.

R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1989.

T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al. HuggingFace's Transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

W. Xiao and G. Carenini. Extractive summarization of long documents by combining global and local context. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing*, 2019.

Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics*, 2016.

J. Zhang, Y. Zhao, M. Saleh, and P. J. Liu. PEGASUS: Pre-training with extracted gap-sentences for abstractive summarization. *arXiv preprint arXiv:1912.08777*, 2019.

T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi. BERTScore: Evaluating text generation with BERT. In *Proceedings of the International Conference on Learning Representations*, 2020.

Appendix A

Example Summaries

In this appendix we display some examples of summaries generated in our experiments, in comparison with their respective reference summaries and the document used to generate them. All samples displayed are obtained from the TripAdvisor dataset. We present the best case and worst case for each experiment, determined by the corresponding BERTScore.

A.1 Extractive Summarization

A.1.1 DistilBERT baseline

Best case

Reference summary 1: A girl and her mother were heading to NYC for the first time. They wanted to know must see christmas places in NYC. She knew about the Rockefeller Center Christmas tree. The suggested must see christmas places were The Department Store windows, Macys, Saks, Bloomingdales, Lord and Taylor, the Xmas Markets which are all over, the ice-rink in Central park at dusk, when all the buildings round the park are lit up. Coming across Xmas trees in certain areas, ie Wall Street, Lincoln Ctr. The other suggestions were the Radio City Christmas Spectacular is great. It's aimed at kids. The rockefeller tree and ice rink is a must see and there are some nice restaurants overlooking the ice rink for dinner and lunch. Central park is lovely in winter, Grand Central Station there was a lovely Christmas Market in the Vanderbuilt Hall which is worth a visit. And of course the window displays on 5th Avenue and Macys are great. Someone suggested them to see the Staten Island Ferry ride in the early evening when it is just darks and the skyline buildings are lit up for Christmas.
Reference summary 2: The original poster were going to NYC during 30/11 – 4/12 with her mom. She wanted to know what are the Christmas "must see" places. People suggested numerous places, notably Rockefeller Center Christmas tree and ice-rink in Central Park.
Extracted summary: My Mum and i are heading to NYC for the 1st time in a few weeks. We are there 30th Nov - 4th Dec. The Department Store windows, Macys, Saks, Bloomingdales, Lord and Taylor, the Xmas Markets which are all over (many have the same items), the ice-rink in Central park at dusk, when

all the buildings round the park are lit up. Coming across Xmas trees in certain areas, ie Wall Street, Lincoln Ctr. My mum and I have been to New York at Christmas time every year for the last 4 years and love it! The Radio City Christmas Spectacular is great. We've seen that 3 times now so this year we have opted for White Christmas at the Marquis theatre instead. The rockefeller tree and ice rink is a must see and there are some nice restaurants overlooking the ice rink for dinner and lunch. if you go to see Grand Central Station there is a lovely Christmas Market in the Vanderbuilt Hall which is worth a visit. What time does the MKt in central station go on to? My personal must is a Staten Island Ferry ride in the early evening when it is just darks and the skyline buildings are lit up for Christmas. Also check out the Neopolitan christmas tree at Met Museum and consider a visit to the Cloisters, the Met's medieval art collection in Washington Heights which is decorated for Xmas,and sometimes has live holiday music. Worth it

Worst case

Reference summary 1: A person initiated discussins about new york city subway station agents. He was wondering if a private sector player could hire people who know their way around subway system and would like to help strangers. Another poster suggested that there are a couple of iphone apps like citytransit and exitstrategy for this purpose.
Reference summary 2: A person posted a newspaper article about new york city subway station agents. He then raised a question if a private sector player could start a business by hiring people who know their way around subway system and would like to help other people completely stranger to them. Another user said that a couple of iphone apps like citytransit and exitstrategy already provide this service.
Extracted summary: From today's TImes: ————————. Afzal Hossain is the New York City subway's newest customer assistant, and he is not happy about it. After nearly 300 station agents were removed from the system late last month because of hard fiscal times, the task of guiding passengers and monitoring late-night riders has fallen to unsuspecting Samaritans like Mr. Hossain, a snack vendor in the Times Square station. ————————————————————————————. . But where could they find a pool of people who know their way around the subway, and who actually seem to ENJOY giving advice to to total strangers???? Just sayin' "For a fee, Ugarte." Oops! These young ladies were already ahead of the curve - so to speak - in April, adn MTA lawyers put the kibosh on them. gothamist.com/2008/04/09/coffeeteaor_the.php "One passenger, 44-year-old Steven Faria, told the Post, "With these girls, I'd be on the train all day. LOL, QB! And another thing.

A.1.2 T5 Classifier

Best case

Reference summary 1: The person and his/her partner were thinking of hiring a car for 5.5 days from Buffalo and returning it to NYC.They wanted to spend a couple of days in Toronto so wanted to more about border crossing as they had australian passports/licenses. They planned to pick up the car in

Buffalo on the 28th of Dec and returning it in NYC on the 2nd of Jan. They had some questions like what were the road conditions generally like that time year,do they need to carry chains? or was that something the car hire company would provide? Also The person noticed when enquiring about rental cars online there were additional options inc. loss damage waiver, supplemental liability etc. He/she was confused as to what kind of insurance was included and what was optional. The person was advised to should check with the car rental company about their policy regarding the border-crossing into Canada and was suggested to look in the forum as other people has asked that question. The weather was quite variable at that time of year. The roads may be perfectly dry, or may have a lot of snow. In any case, no one uses chains, and in fact he/she thought they were illegal these days because of the damage they do to the roads. A rental car will be equipped with all-weather tires, and that's all he/she was needed. Even if it's quite snowy, the road crews were experienced in keeping the highways plowed and salted. Just maintain a safe speed. If the weather was very bad, he/she might need to postpone the trip if he/she was not experienced with driving on snow or ice. Also, be sure to take a travel kit with energy bars, some bottled water, a blanket, and a fully charged cell phone against the (unlikely) possibility that he/she would get stranded. Generally, American drivers who have insurance on their own vehicles were covered for any damage to the rental vehicle and do not need to buy the supplemental insurance offered by the rental companies but he/she was not sure whether that's the case for someone coming from another country. A person should check with his/her insurance company in the U.K. about their policy. At any rate, the charge for the supplemental insurance wasn't too great. Finally, because he/she was traveling over the holidays, do be especially careful on the roads and try to avoid traveling late at night if at all possible. He/she was going to do a lot of driving.The trip from Buffalo to New York City was about seven hours, so, especially if the weather was bad, allow oneself plenty of time for slowdowns due to holiday traffic, weather delays, etc., especially if he/she have a flight to catch in NYC

Reference summary 2: A person wanted to know if she'll be faced with any drama crossing the US/-Canada borders, since she'll be traveling on Australian passports/licenses and in rental car. She also wanted to know the road conditions a this time of the year and wondered if she has to carry chains or would a car hire company provide those. She's also confused about the kinds of insurances she has to choose when renting a car. A person responded that she should check with the car rental company about their policy about border-crossing into Canada. She also said that the weather is quite uncertain, and no one uses chains because they are illegal. As for the insurance, she said that the original poster should check with her insurance company back at home.

Extracted summary: Hi Everybody, My partner and I are thinking of hiring a car for 5.5 days from Buffalo and returning it to NYC. We are wanting to spend a couple of days in Toronto, but we are wondering if we'll have any dramas crossing the borders (we will be traveling on Australian passports/licenses). We plan to pick up the car in Buffalo on the 28th of Dec and returning it in NYC on the 2nd of Jan. I'm confused as to what kind of insurance is included and what is optional... any help would be much appreciated. You should check with the car rental company about their policy regarding the border-crossing into Canada. Our weather is quite variable at that time of year. The roads may be perfectly dry, or we may have a lot of snow. In any case, no one uses chains, and in fact I think they are illegal these

days because of the damage they do to the roads. Your rental car will be equipped with all-weather tires, and that's all you'll need. Just maintain a safe speed. Also, be sure to take a travel kit with energy bars, some bottled water, a blanket, and a fully charged cell phone against the (unlikely) possibility that you'll get stranded. Generally, American drivers who have insurance on their own vehicles are covered for any damage to the rental vehicle and do not need to buy the supplemental insurance offered by the rental companies – but I'm not sure whether that's the case for someone coming from another country. You should check with your insurance company in the U.K. about their policy.

Worst case

Reference summary 1: A person wrote a detailed 5 day New York trip report. It covered detail such as hotels, food, places to visit, subway and stores. He had a wonderful trip.
Reference summary 2: The original poster shared his 5-day NYC trip report where he spent most time in LSE and SOHO, and mid-town.
Extracted summary: We had a late lunch at White Slab Palace...a funky joint on the LES with a Scandinavian theme. We grabbed a pre-dinner drink at The Back Room, a speakeasy/prohibition themed bar with complicated access and drinks served in teacups. Then we had dinner at the hotel restaurant Shang to claim our $100 dinner credit (part of the amazing hotel deal). After brunch, walked over to Lincoln Center to see South Pacific....very enjoyable. We had dinner at Mercer Kitchen....good food, decent service, but not exceptionally memorable. Hit the subway back to the LES to a reservation at Milk & Honey (small bar that focuses on specialty drinks). Day 4 - Lunch at Macelleria in the Meatpacking District. The trick to M&H was that we went on a Monday night. I think that's the only way to get the 10:30 reservation time that we were able to get. Otherwise, we would have had to try for a much earlier time. Yes, my wife is from Sweden. Wow, what a great room rate! Great food too.

A.1.3 T5 Hybrid

Best case

Reference summary 1: KLMTrinidad needed advise for the best way to get from JFK to New Rochelle using public transport. He/she knew that the taxi would cost $80-85 USD which was quite expensive. Crans suggested taking a train as New Rochelle was on the New Haven line of the Metro-North Railroad. He/she was also suggested to a bus from JFK to Grand Central Station, where one can board the Metro-North train. Bilmin09 suggested taking a rental car at JFK and driving to New Rochelle because it would be useful for transportation in New Rochelle and would cost same as the taxi fare. KLMTrinidad would be attending a wedding in New Rochelle and he/she wanted to know whether it would be easy to take taxi from the station. Blue Bird Taxi -914-632-0909 could be an option. KLMTrinidad needed information about the taxi charge from the station to the Marriott Residence Inn. CockleCove informed that the combined cost of getting from JFK to New Rochelle and back to JFK – using the Airport Express Service bus and MetroNorth – would be $40-45 round-trip, depending on whether KLMTrinidad was riding the MetroNorth train at times it deems to be peak or off-peak. The New Rochelle train station was only 1/2

mile from the Residence Inn, and according to MetroNorth's page re: the New Rochelle station, Bluebird Taxi has its office right at the train station. KLMTrinidad also wanted to know whether there would be a lot of stairs.

Reference summary 2: A person asked for advices on the best public transportation option from JFK to New Rochelle. A few people mentioned Metro-North train + taking Bluebird Taxi from train station to the hotel. The hotel is not far from the train station so the fare should not be much.

Extracted summary: Can someone please advise the best way to get from JFK to New Rochelle using public transport? I know that by Taxi it is $80-85 USD which is quite expensive. New Rochelle is on the New Haven line of the Metro-North Railroad: http://www.mta.info/mnr/html/mnrmap.htm (It's the red line.) In that case you may want to rent a car at Jfk and drive. The one day cost would be equal to the taxi fare and you will be in control. I will be attending a wedding in New Rochelle. When I take the Metro-North to New Rochelle, would it be easy for me to come out of the station and get a taxi to a hotel? Public transportation is fine in a big city but limited outside of the city. You need to think through the entire time there to determine your best way of getting around. You may contact the hotel, sometimes they have the info on the best way to get directly there from airports. How much would the taxi charge from the station to the Marriott Residence Inn? Your combined cost of getting from JFK to New Rochelle and back to JFK – using the Airport Express Service bus and MetroNorth – will be $40-45 round-trip, depending on whether you'll riding the MetroNorth train at times it deems to be peak or off-peak. The New Rochelle train station is only 1/2 mile from the Residence Inn, and according to MetroNorth's page re: the New Rochelle station, Bluebird Taxi has its office right at the train station.

Worst case

Reference summary 1: A person wrote a detailed 5 day New York trip report. It covered detail such as hotels, food, places to visit, subway and stores. He had a wonderful trip.

Reference summary 2: The original poster shared his 5-day NYC trip report where he spent most time in LSE and SOHO, and mid-town.

Extracted summary: Day 1 - arrived early on a red eye. They really didn't have to do this: we had an amazing rate at $167/night for a King studio suite, and I really should have booked the extra night to ensure the early access. We had a late lunch at White Slab Palace...a funky joint on the LES with a Scandinavian theme. We grabbed a pre-dinner drink at The Back Room, a speakeasy/prohibition themed bar with complicated access and drinks served in teacups. After brunch, walked over to Lincoln Center to see South Pacific....very enjoyable. Day 3 - Took the subway up to mid-town to walk around and check out the stores. Hit the subway back to the LES to a reservation at Milk & Honey (small bar that focuses on specialty drinks). Then we walked over to Arlene's Grocery to check out karaoke night (very entertaining). Day 4 - Lunch at Macelleria in the Meatpacking District. (very tasty) Walked around a bit, checked out The Standard and some of the shops before heading uptown. Hopefully we can make Milk and Honey this time around. I think that's the only way to get the 10:30 reservation time that we were able to get. Wow, what a great room rate!

A.2 Abstractive Summarization

A.2.1 T5 Summarizer

Best case

Reference summary 1: A person wanted to know that whether Kyotofu is just a sit-down restaurant type or to go restaurant. Someone replied that Kyoto-fu hosts events for both smaller parties and larger affairs, from 5 to 65 people, in x modern and inviting restaurant. It offer unique and flexible menu choices for both sit-down dinners and passed cocktail parties. Kyotofu also offers a full bar or a selection of beverages based upon your budget and preferences. From business luncheons and bridal showers to private sake tastings, corporate gifts and wedding cakes, our in-house team of professional party planners can cater to customer every need.

Reference summary 2: A person wanted to know if Kyotofu is a sit-down, restaurant-type place or a go in and get a cupcake to go. Most people seem to think that it is more of a sit-down place , but it does offer take-out.

Generated summary: Kyotofu hosts events for both smaller parties and larger affairs, from 5 to 65 people, in our modern and inviting restaurant. We offer unique and flexible menu choices for both sit-down dinners and passed cocktail parties. We also offer a full bar or a selection of beverages based upon your budget and preferences. From business luncheons to private sake tastings, corporate gifts and wedding cakes, our in-house team of professional party planners can cater to your every need.

Worst case

Reference summary 1: A person wrote a detailed 5 day New York trip report. It covered detail such as hotels, food, places to visit, subway and stores. He had a wonderful trip.

Reference summary 2: The original poster shared his 5-day NYC trip report where he spent most time in LSE and SOHO, and mid-town.

Generated summary: thankfully the Thompson LES gave us our room at 6:30 a.m. They really didn't have to do this: they had an amazing rate at $167/night for a King studio suite, and I really should have booked the extra night to ensure the early access. After a short nap, we hit the streets and walked around the LES and Soho. We grabbed a pre-dinner drink at The Back Room, a speakeasy/prohibition themed bar with complicated access and drinks served in teacups. Day 2 - Slept in...first activity was brunch at Mercer Kitchen...good food, decent service, but not exceptionally memorable. It was pretty chilly by the water. We had a great time at the hotel restaurant Shang to claim our $100 dinner credit, and then went back to Greenwich Village to see South Pacific...a funky joint on the High Line, and was able to make Milk and Honey this time around. Then we had dinner at White S

A.2.2 T5 Summarizer after Extractive

Best case

Reference summary 1: There was a special helicopter deal for $59 for Continental Airline passengers arriving at/departing from Newark Liberty International Airport. The offer was till Oct 30. The online link of this deal did not allow people to find out more details. Someone suggested that the terms and conditions were as follows the promotional fare does not include passenger facility charges of up to $18 per segment, Sept. 11th security fee of up to $5 each way, Federal segment tax of $3.60 per segment or 7.5% U.S. Transportation Tax. Someone informed that the federal government considers the helicopter ride to be a flight so there were a bunch of mandatory surcharges but it was still cheaper than the helicopter sighteeing tours of the same length.
Reference summary 2: A person shared that there's a special deal for Continental Airline passengers arriving at or departing from Newark Liberty International Airport from now until October 30 to use the US Helicopter service to get from and to EWR for just $59.
Generated summary: special deal for Continental Airline passengers arriving at/departing from Newark Liberty International Airport from now until October 30 . it does not include passenger facility charges of up to $18 per segment, Sept. 11th security fee of upto $5 each way, Federal segment tax of $3.60 per segment or 7.5% U.S. Transportation Tax . if you haven't yet purchased your ticket with Continental Airlines, book now .

Worst case

Reference summary 1: A person was going to NYC to see Keith Urban at the Garden and needed suggestions to eat economical food in the area.Someone suggested him to go 9th avenue around 50th street there were many restaurants.The other suggestion was for Korean food along 32nd street.The suggested restaurants were Tick Tock Diner, Skyline Dinner, Tir Na Nog , Molly Wee pub, Mustang Sally Local 33. The person finally went to the Molly Wee and enjoyed the food.
Reference summary 2: A person was going to Keith Urban concert at Madison Square Garden and wanted suggestions on cheap eat in the area. One person suggested to look on 9th Ave. around 50th St. which is not near the Garden but can catch the subway to the Garden easily. Another person suggested Korean restaurants along 32rd St. The original poster finally went to the Molly Wee which he liked.
Generated summary: Keith Urban is a 'country music' singer . he's married to Nicole Kidman . she's a big fan of country music in the USA .

A.2.3 T5 Hybrid

Best case

Reference summary 1: Powerade was sponsoring a free water taxi/bus shuttle together with NY Watertaxi to the US Open. The passenger have to make a reservation. Someone rode the Water taxi. It went very smoothly and was a fun way to get to the US Open. It was a beautiful Saturday morning. The

subway was very crowded and took about the same amount of time. The water taxi was very relaxing and a fun way to see the skyline. There was no line to get on and the buses were ready to take us to the Open. The ferry took 35 to 45 minutes.'

Reference summary 2: A person shared a cool way to get to the US Open, which is a Powerade-sponsored free water taxi/bus shuttle, and suggested that reservations seem to be going fast. The ferry leaves from and returns to the E. 35th St. Marina and drops off at the Worldś Fair Marina. Upon arriving at the World's Fair Marina, buses will transport fans directly to the tennis gates. Buses will be waiting to return fans to the World's Fair Marina for the ferry ride back to the E. 35th St Marina all FREE of charge. One person asked where E. 35 St. Marina is and how to get there from Affina Dumont. She also wanted to know how long the water taxi ride would take. She was told that she could get the walking direction from Google maps by entering "affinia dumont nyc to east 35 th st. marina nyc" and that the ferry ride takes 35-45 minutes, while the shuttle bus takes about 10 minutes.

Generated summary: POWERADE ZERO and Venus Williams are sponsoring a free water taxi/bus shuttle to the US Open . the ferry leaves from and returns to the E. 35th St. Marina, and drops off at the World's Fair Marina. Upon arriving, buses will transport fans directly to the tennis gates. the breeze and spectacular city views are also FREE of charge. the shuttle bus takes about 10 minutes max.

Worst case

Reference summary 1: A group of people were shocked on the closure of the Cafe des Artistes. The place had historic "old time" setting and good food. It was a favorite restaurant for some people. John Dory and Town & Country were also the closed restaurant.

Reference summary 2: A person shared that Cafe des Artistes has closed, and that she had some lovely dinners there and was sorry to see it go. Some people were also sad to see it go, but one person pointed out that she wasn't shocked because, in recent years, the fare at the restaurant was too erratic in quality. Others also shared that John Dory and Town closed as well.

Generated summary: a new post-recessionary "normal" we live in - your restaurant better be 1. beloved 2. in a really accessible/frequently visited neighborhood 3. have lots of pedestrian traffic 4. have a good walk-in trade 4. be un-stuffy and casual 5. have market driven menus John Dory is looking for better space. I'm sure it'll be back.