TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

LIST OF ABREVIATIONS

BBC	British Broadcasting Corporation
CE	Cross-Entropy
CNN	Convolutional Neural Network
CRM	Client Relationship Management
DL	Deep Learning
DUC	Document Understanding Conference
EOS	End Of Sequence
FFNN	Feed-Forward Neural Network
GAN	Generative Adversarial Network
GDELT	Global Database of Events, Language and Tone
LCS	Longest Common Subsequence
LM	Language Model
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	MultiLayer Perceptron
MSE	Mean Squared Error
NIST	National Institute of Standards and Technology
NLG	Natural Language Generation
NLP	Natural Language Processing

PE	Positional Encoding
R&D	Research & Development
ReLU	Rectified Linear Units
RNN	Recurrent Neural Network
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SOTA	State-Of-The-Art
TL;DR	Too Long; Didn't Read
WIIM	Why Is It Moving

LIST OF SYMBOLS

a	A scalar or string of characters
$\boldsymbol{a}$	A vector or sequence
$\boldsymbol{A}$	A matrix
$\mathbb{A}$	A set
$\mathbb{X}$	A set of training examples
$\boldsymbol{x}^{(i)}$ or $\boldsymbol{X}^{(i)}$	The i-th item (input) from a dataset
$y^{(i)}$, $\boldsymbol{y}^{(i)}$ or $\boldsymbol{Y}^{(i)}$	The target associated with $\boldsymbol{x}^{(i)}$ or $\boldsymbol{X}^{(i)}$

INTRODUCTION

The Canadian company Croesus offers a dashboard for financial advisors. Their flagship product, a software called Croesus Advisor, brings together investment portfolio[1] management and Customer Relationship Management (CRM) functionalities.

In an effort to provide its users with the most advanced and useful tools as possible, the company launched an R&D project whose aim is to automate the generation of reports explaining the performance of investment portfolios over a given period of time. The generated reports will need to inform their readers about the factors that influenced the performance of the portfolio, the degree of their influence and the extent to which these factors deviate from the norm. This type of data-to-text system falls under the domain of Natural Language Generation (NLG). Figure 0.1 presents a comparison between a generic data-to-text pipeline architecture (Reiter, 2007; Reiter & Dale, 2000) and the architecture used in this project.

The five stages of a generic data-to-text pipeline architecture are:

1. **Signal analysis**: analyzing numerical and other input data, looking for patterns and trends;
2. **Data interpretation**: identifying more complex messages from the patterns and trends detected during signal analysis;
3. **Content determination**: deciding which messages should be mentioned in the generated text;
4. **Document structuring**: creating a document and rhetorical structure around the selected messages;
5. **Microplanning and realisation**: creating an actual text which communicates the document plan.

[1] An investment portfolio is a set of financial assets, such as company shares, bonds, cash, etc.

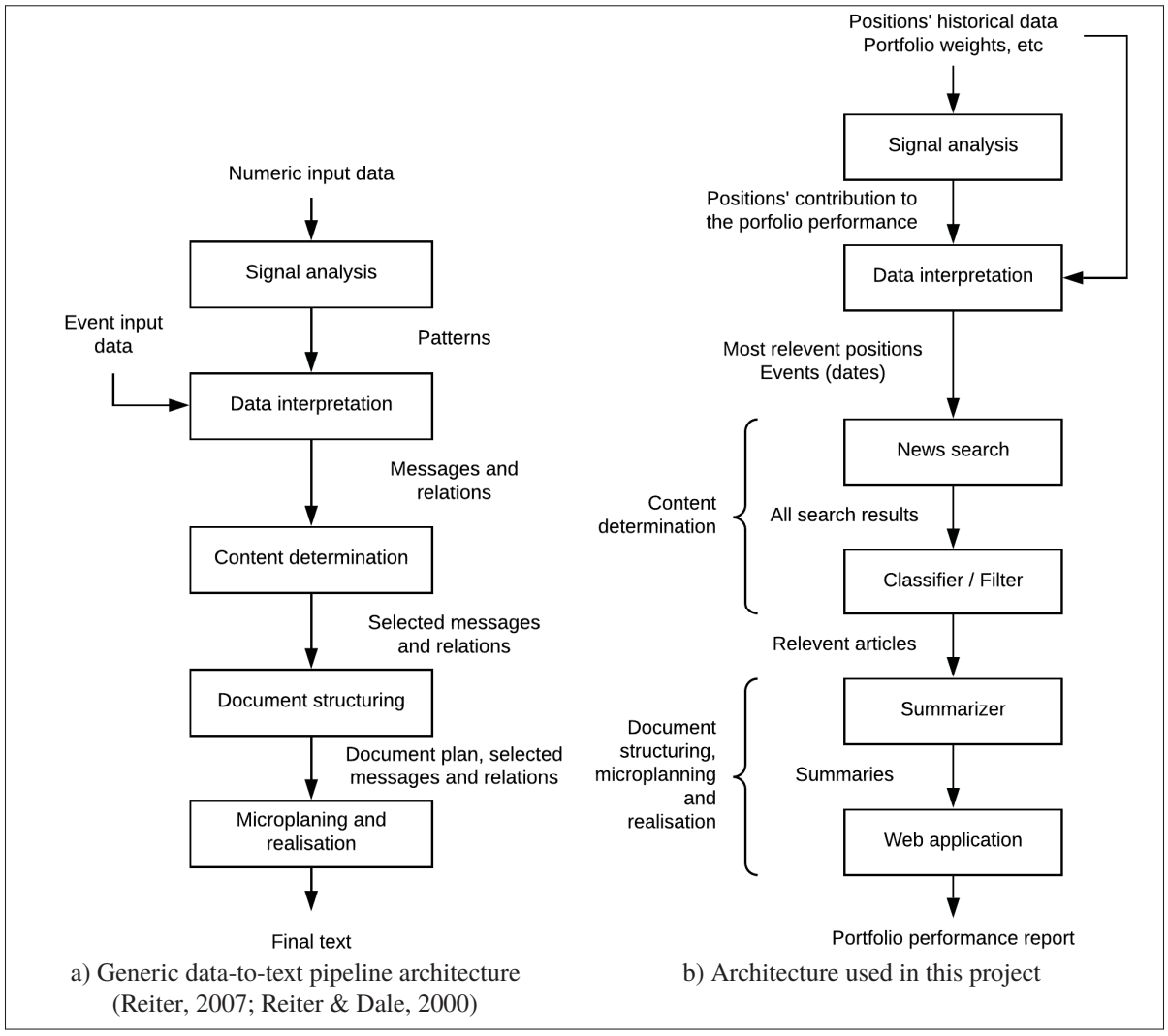

Figure 0.1 Comparison between a generic data-to-text pipeline architecture and the architecture used in this project

Figure 0.2 gives an example of typical input data for the project's pipeline. Also, it is worth noting that the documents returned by the News search component mainly consist of financial news articles. However, since the experiments in this work focus exclusively on the Summarizer component, standard datasets of generic news are used instead. Table 0.1 provides an example of the desired output of the project's pipeline.

Not all stages of the project's pipeline constitute a research problem. For example, the contribution of individual assets to the portfolio's overall performance (signal analysis) will be determined using well known techniques from the field of finance (Lawton & Jankowski, 2009). Likewise, document structuring and microplaning will be implemented using current Web design patterns (Marcotte, 2011).

Figure 0.2 Example of typical input data for the project's pipeline: the value of a Pfizer Inc. share (PFE) through 2019. Source: https://ca.finance.yahoo.com/

However, multiple research problems remains. First, the time series describing the positions that most significantly contributed to the performance of a portfolio must be analyzed in order to identify "significant events". This corresponds to the data interpretation stage of the pipeline. Next, documents describing these events must be retrieved among multiple documents which may, or may not be relevant. This corresponds to the content determination stage of the pipeline. Then, the sequence of relevant documents describing significant events needs to be summarized in a coherent and informative narrative of controllable length. This encompass the document structuring, microplanning and realisation stages of the pipeline. Finally, the generated reports should be available in English and French to comply with Canadian laws and customs. This last requirement applies to the entirety of the reporting pipeline.

Table 0.1 Example of the desired output of the project's pipeline: a timeline of event summaries. Source: https://www.benzinga.com/apis/cloud-product/bz-why-is-it-moving/

Date	Event Summary
2019-01-28	Pfizer shares are trading lower after the company said it expects 2019 EPS of $2.82 to $2.92 versus the $3.04 analyst estimate.
2019-04-16	Shares of several drug companies are trading lower. Weakness may be tied to uncertainty over healthcare policy going into the 2020 primaries and Washington's recent efforts to curb prescription drug prices.
2019-04-30	Pfizer shares are trading higher after the company reported better-than expected Q1 earnings and raised guidance.
2019-08-05	Shares of several drug/pharma/medical equipment companies trading lower given weakness in broader stock market following concerns with China trade tensions.
...	...

The objective of the R&D effort is to evaluate the feasibility of the whole project. A positive outcome followed by a successful implementation and marketing campaign could result in a larger market share and higher revenues for Croesus.

This work constitute the first step in this endeavour.

Efforts initially focused on solving the data interpretation problem using an anomaly detection strategy. However, this approach was quickly abandoned. As it turns out, defining the concept of "significant event" in the context of automatic portfolio performance evaluation is a complex task requiring the involvement of domain experts which were not available at the moment. Efforts subsequently focused on assessing the feasibility of another high risk stage of the pipeline: microplanning and realisation using automatic summarization. This choice was made because experiments in this field could be conducted using easily available training and evaluation datasets and did not require finance expertise.

Contributions of this work include: the design of the project's data-to-text architecture (figure 0.1), the successful application of output control mechanisms to the task of abstractive summarization and the mitigation of the risk with respect to unsupervised automatic summarization.

The rest of this document is organized as follows: chapter 1 describes the background necessary to the understanding of the proposed models; chapter 2 describes the methodology used to conduct experiments; Finally, chapters 3 and 4 introduces the experiments themselves with accompanying results.

CHAPTER 1

BACKGROUND

This chapter describes the background information required to understand the challenges that the R&D project at hand offer and the models proposed to overcome them.

1.1 Automatic Summarization

Automatic summarization is concerned with compressing document(s) into a concise and fluent summary while preserving the most important information. There are many approaches to the automatic generation of summaries. One of the most important distinction to be made is whether the summary generation system is developed using a supervised approach, that is using examples of documents with their corresponding summaries, or without supervision, that is without such examples. Although supervised approaches recently made great progress (Lewis, Liu, Goyal, Ghazvininejad, Mohamed, Levy, Stoyanov & Zettlemoyer, 2019; Liu & Lapata, 2019; Nallapati, Zhou, Santos, Gulcehre & Xiang, 2016; Rush, Chopra & Weston, 2015; See, Liu & Manning, 2016; Zhang, Zhao, Saleh & Liu, 2019), they are not without their shortcomings when it comes to real-world applications. One of their main drawbacks is the fact that currently available datasets are rarely an optimal fit for specific application domains. Another limitation is the lack of datasets in languages other than English. Collecting and annotating large amount of aligned data require great efforts, therefore, the ability to train summarization models in an unsupervised fashion is interesting because it eliminates the need to provide reference summaries. For these reasons, this work focuses on the unsupervised training of automatic summarization systems.

Summarization algorithms can be split into two main categories: extractive and abstractive. Extractive algorithms (Erkan & Radev, 2004; Zheng & Lapata, 2019) redact summaries by concatenating relevant portions of the input, while abstractive algorithms (Liu & Lapata, 2019; Rush *et al.*, 2015) generate new texts that may use terms that are not present in the input (Das & Martins, 2007; Nenkova & McKeown, 2011). It has been observed that human written summaries tend to be abstractive (Kryściński, Keskar, McCann, Xiong & Socher, 2019).

1.2 Artificial Neural Networks

Inspired by the mammal brain, Artificial Neural Networks (ANN) have proven themselves powerful tools to deal with multiple Artificial Intelligence (AI) tasks in recent years. This section offers an introduction to the most common types of ANNs currently used in the field of Natural Language Processing (NLP).

Artificial Neural Networks are a vast and ever-evolving subject and for this reason an in depth survey is out of the scope of this work. Readers interested in the matter are invited to consult one of the many resources on the subject, such as the excellent *Deep Learning* textbook (Goodfellow, Bengio & Courville, 2016).

1.2.1 Feed-Forward Neural Networks

One of the reasons why ANNs are said to be inspired by the mammal brain is that their core unit, the Perceptron (Rosenblatt, 1958), was originally modeled after neuron cells. The Perceptron is a Machine Learning algorithm for the modeling of binary classifiers. A linear function in itself, it uses a set of learned weights $\boldsymbol{w} = [w_0, ..., w_n]$ and an activation function $g(\cdot)$ to decide whether or not an input $\boldsymbol{x} = [x_0, ..., x_n]$ belongs to a specific class. If $\boldsymbol{w}$ and $\boldsymbol{x}$ are represented by single column matrices, then a Perceptron's output y is computed as follow:

$$y = g(\boldsymbol{w}^T \boldsymbol{x}) \tag{1.1}$$

where $g(\cdot)$ is an activation function such as the sigmoid function:

$$\sigma(v) = \frac{1}{1 + e^{-v}} \tag{1.2}$$

the hyperbolic tangent function:

$$\tanh(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}} \tag{1.3}$$

or Rectified Linear Units (ReLU) function:

$$\text{ReLU}(v) = \begin{cases} v & \text{if } v > 0 \\ 0 & \text{otherwise} \end{cases} \tag{1.4}$$

Figure 1.1 illustrates the architecture of the Perceptron.

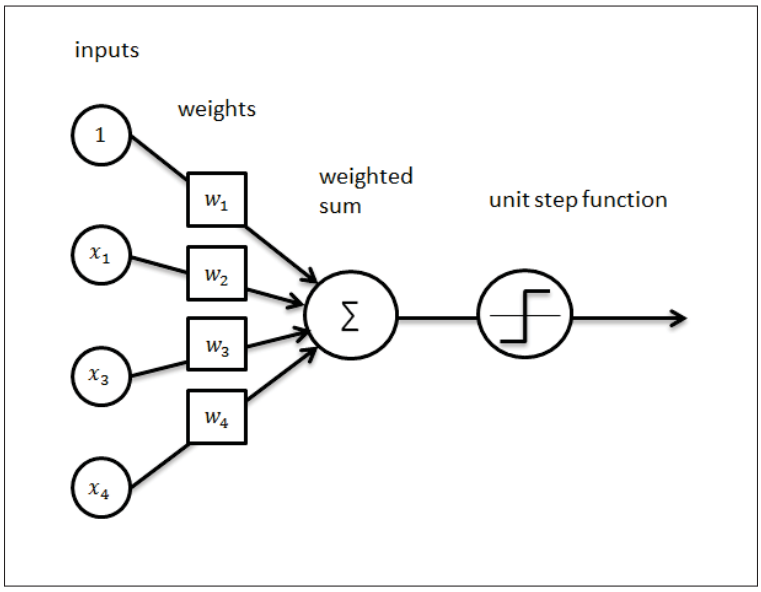

Figure 1.1 Architecture of a Perceptron. Source: http://ataspinar.com/2016/12/22/the-perceptron/

Since they can only learn linear functions, Perceptrons are not powerful enough to model many complex, real-life tasks. However, this limitation can be overcomed by combining multiple layers of Perceptrons into an Artificial Neural Network architecture named Feed-Forward Neural Network (FFNN). Figure 1.2 illustrates the general architecture of such networks. FFNNs can be understood as a composition of different functions, for example as the first (input) layer $f^{(1)}$, the second (hidden) layer $f^{(2)}$ and the third (output) layer $f^{(3)}$ being chained into a $f(x)$ function representing the whole FFNN, so that $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. It is this stacking of functions that allows Feed-Forward Neural Networks, and Artificial Neural Networks in general, to model very complex tasks.

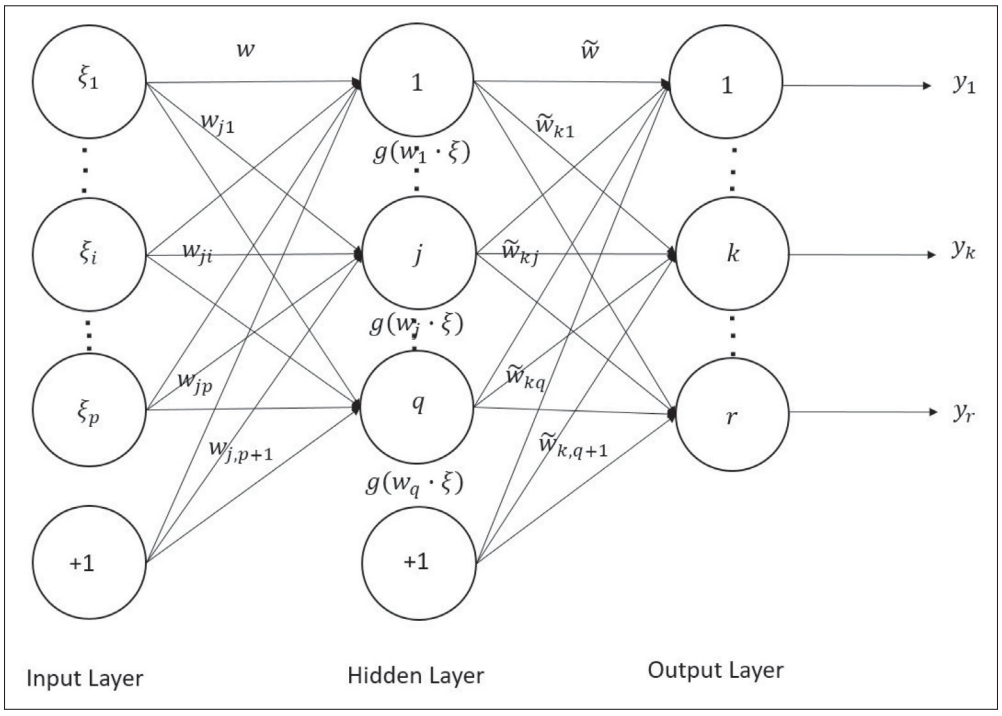

Figure 1.2 General architecture of a Feed-Forward Neural Network (FFNN). The first and last layers of a FFNN are named input and ouput layers, respectively. Any layer in between are labeled hidden layers. Source: Alemu *et al.* (2018)

When training an Artificial Neural Network, its weights are first initialized with small random values and subsequently updated using training data and an iterative optimization technique such as gradient descent (Cauchy, 1847). The training dataset needed to optimize a FFNN is a collection of "correct" (input, output) tuples for a given problem[1].

When using the gradient descent optimization algorithm, every step of the process starts by prompting the FFNN to produce an output for every input from the training dataset. The set of resulting outputs $\hat{\mathbb{Y}} = \{\hat{\boldsymbol{y}}^{(0)}, \ldots, \hat{\boldsymbol{y}}^{(k)}\}$ is then compared to the set of correct outputs from the dataset $\mathbb{Y} = \{\boldsymbol{y}^{(0)}, \ldots, \boldsymbol{y}^{(k)}\}$ so a measure of loss J (also referred to as cost, or error) can be computed. Mean Squared Error (MSE) is a common choice of loss function for real-value

[1] In practice, a training dataset almost always contain noise of one form or another.

outputs:

$$J_{MSE}(\mathbb{Y}, \hat{\mathbb{Y}}) = \frac{1}{k} \sum_{i=0}^{k} \left(\boldsymbol{y}^{(i)} - \hat{\boldsymbol{y}}^{(i)} \right)^2 \tag{1.5}$$

Cross-Entropy (CE) is a common choice of loss function for binary output values:

$$J_{CE}(\mathbb{Y}, \hat{\mathbb{Y}}) = -\frac{1}{k} \sum_{i=0}^{k} \left(\boldsymbol{y}^{(i)} \log \hat{\boldsymbol{y}}^{(i)} + (1 - \boldsymbol{y}^{(i)}) \log (1 - \hat{\boldsymbol{y}}^{(i)}) \right) \tag{1.6}$$

The backpropagation algorithm[2] (Rumelhart, Hinton & Williams, 1986) is then used to compute the gradient of J with respect to the weights $\boldsymbol{w}$ of the FFNN, denoted $\nabla_{\boldsymbol{w}} J(\boldsymbol{w})$. The gradient is useful to the optimization of the Artificial Neural Network because it tells the direction in which updating its weights $\boldsymbol{w}$ produces the fastest increase of error J. Accordingly, updating $\boldsymbol{w}$ in the opposite direction of the gradient also produces the fastest decrease of J. For this reason, every step of a gradient descent updates $\boldsymbol{w}$ as follow: $\boldsymbol{w}_{n+1} = \boldsymbol{w}_n - \alpha \nabla_{\boldsymbol{w}} J(\boldsymbol{w}_n)$, where α is an hyper-parameter named *learning rate*. Given that α is small enough, iteratively executing gradient descent steps will move J until the nearest local minimum is finally reached. See figure 1.3 for a visual depiction of the complete algorithm.

The necessity to generate an output for every item in the training dataset in order to execute a single gradient descent step is often challenging due to the ever-growing size of modern datasets and current hardware limitations. For this reason, the vanilla gradient descent algorithm is generally discarded in favor of other variants of the algorithm, such as stochastic gradient descent and mini-batch gradient descent. Using stochastic gradient descent, weights $\boldsymbol{w}$ are updated after each training sample is used. Using mini-batch gradient descent, weights $\boldsymbol{w}$ are updated after each k training samples have been used.

---

[2] The backpropagation algorithm mainly consists of a recursive application of the chain rule of derivatives.

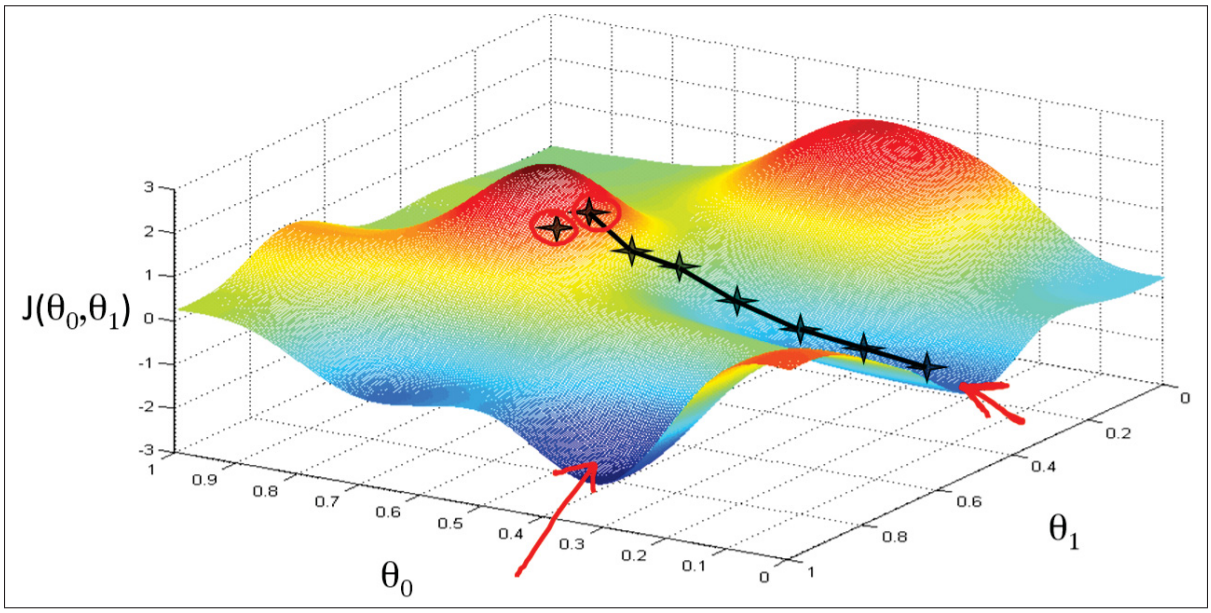

Figure 1.3 Visualization of a gradient descent procedure. The arrows point at two local minima of loss function J. Note that there is no guarantee that a gradient descent will converge to the global minimum. The exact local minimum that will be reached depends on various factors, such as the model's parameters initial value and the α learning rate in use. Source: https://hackernoon.com/gradient-descent-aynk-7cbe95a778da

1.2.2 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a class of Artificial Neural Networks whose hidden units use recurrent connections to persist information as they go through a sequence of states (Elman, 1990). Persistence of information through time allows RNNs to model temporal dynamic behaviors.

One of the most widely used RNN architecture is Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997). The key concept behind LSTMs is the memory cell. A memory cell is a recurrent unit that persists information in a *cell state* vector and computes an *hidden state* vector at each time step. Memory cells also use *gates* to filter their informational content. LSTMs typically use three types of gates: the *forget* gate, the *input* gate and the *output* gate.

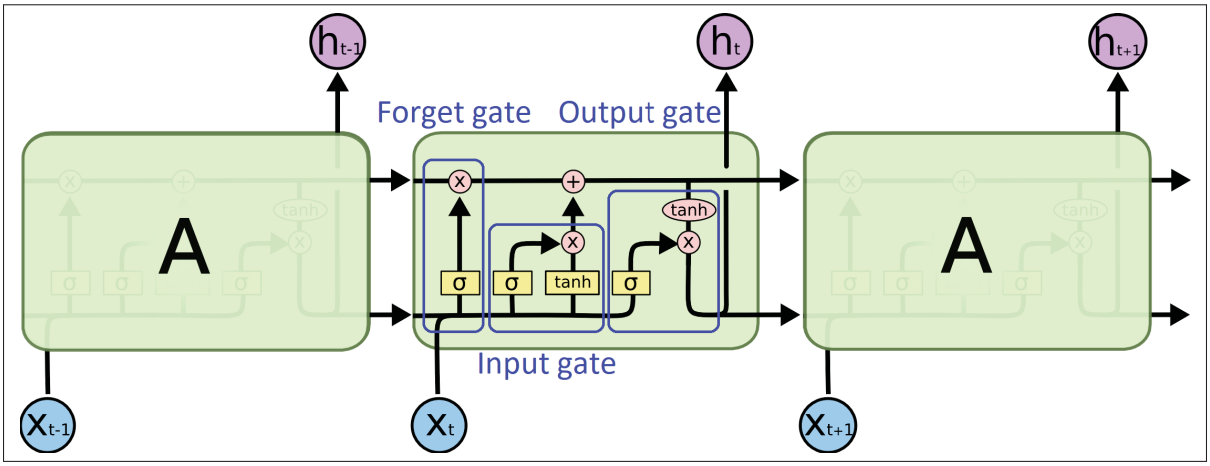

Figure 1.4 Information flow through an LSTM memory cell *A*. Source: https://colah.github.io/posts/2015-08-Understanding-LSTMs

The forget gate's purpose is to filter the information contained in the previous cell state based on the previous hidden state and the current input. The gate takes the form of a single layer FFNN that outputs a filtering vector of the same length as the cell state vector. Each dimension of the filter is a number between 0 and 1: 0 indicates that the corresponding dimension from the previous cell state should be completely forgotten and 1 indicates that it should be completely remembered. Forget gates compute their filters as follow:

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}_f^T(\boldsymbol{h}_{t-1} + \boldsymbol{x}_t)\right) \tag{1.7}$$

where:

- $\boldsymbol{f}_t$: forget filter,
- σ: element-wise sigmoid function (equation 1.2),
- $\boldsymbol{W}_f$: forget gate weights,
- $\boldsymbol{h}_{t-1}$: previous hidden state,
- $\boldsymbol{x}_t$: current input.

Determining what new information should be added to the current cell state involves two steps: generating a vector of *candidate* values and filtering it using the input gate. A single layer FFNN is trained to compute candidate values as follow:

$$\tilde{\boldsymbol{c}}_t = \tanh\left(\boldsymbol{W}_c^T(\boldsymbol{h}_{t-1} + \boldsymbol{x}_t)\right) \tag{1.8}$$

where:

- $\tilde{\boldsymbol{c}}_t$: candidate values,
- tanh: element-wise hyperbolic tangent function (equation 1.3),
- $\boldsymbol{W}_c$: candidate FFNN weights,
- $\boldsymbol{h}_{t-1}$: previous hidden state,
- $\boldsymbol{x}_t$: current input.

Input gate filters are computed as follow:

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}_i^T(\boldsymbol{h}_{t-1} + \boldsymbol{x}_t)\right) \tag{1.9}$$

where:

- $\boldsymbol{i}_t$: input filter,
- σ: element-wise sigmoid function (equation 1.2),
- $\boldsymbol{W}_i$: input gate weights,
- $\boldsymbol{h}_{t-1}$: previous hidden state,
- $\boldsymbol{x}_t$: current input.

The current cell state is obtained by summing the results of applying the forget filter to the previous cell state and the input filter to the current candidate values:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \tag{1.10}$$

where:

- $\boldsymbol{c}_t$: current cell state,
- $\boldsymbol{f}_t$: forget filter,
- $\odot$: element-wise multiplication,
- $\boldsymbol{c}_{t-1}$: previous cell state,
- $\boldsymbol{i}_t$: input filter,
- $\tilde{\boldsymbol{c}}_t$: candidate values.

The output gate's purpose is to filter the information contained in the current cell state based on the previous hidden state and the current input. The gate takes the form of a single layer FFNN that outputs a filtering vector of the same length as the cell state vector. Each dimension of the filter is a number between 0 and 1: 0 indicates that the corresponding dimension from the current cell state should be completely omitted and 1 that it should be completely included. Output gates compute their filters as follow:

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}_o^T(\boldsymbol{h}_{t-1} + \boldsymbol{x}_t)\right) \tag{1.11}$$

where:

- $\boldsymbol{o}_t$: output filter,
- σ: element-wise sigmoid function (equation 1.2),
- $\boldsymbol{W}_o$: output gate weights,
- $\boldsymbol{h}_{t-1}$: previous hidden state,
- $\boldsymbol{x}_t$: current input.

Finally, the current hidden state is computed by applying the output gate filter to the current cell state as follow:

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \tag{1.12}$$

where:

- $\boldsymbol{h}_t$: current hidden state,
- $\boldsymbol{o}_t$: output filter,
- $\odot$: element-wise multiplication,
- tanh: element-wise hyperbolic tangent function (equation 1.3),
- $\boldsymbol{c}_t$: current cell state.

See figure 1.4 for a graphical depiction of the information flow through an LSTM memory cell.

1.2.3 Autoencoders

Modern autoencoder neural network architectures were introduced by (Bengio, Lamblin, Popovici & Larochelle, 2007) and (Ranzato, Poultney, Chopra & Cun, 2007) as a mean to learn "good" representations for initializing deep architectures. A good representation could be defined as a representation potentially useful for addressing tasks of interest. That is a representation

that will help a system quickly achieve higher performance on a task than if it had not learned to form the representation in the first place (Vincent, Larochelle, Lajoie, Bengio & Manzagol, 2010). It is typically expected from a good representation to retain a significant amount of information about the input.

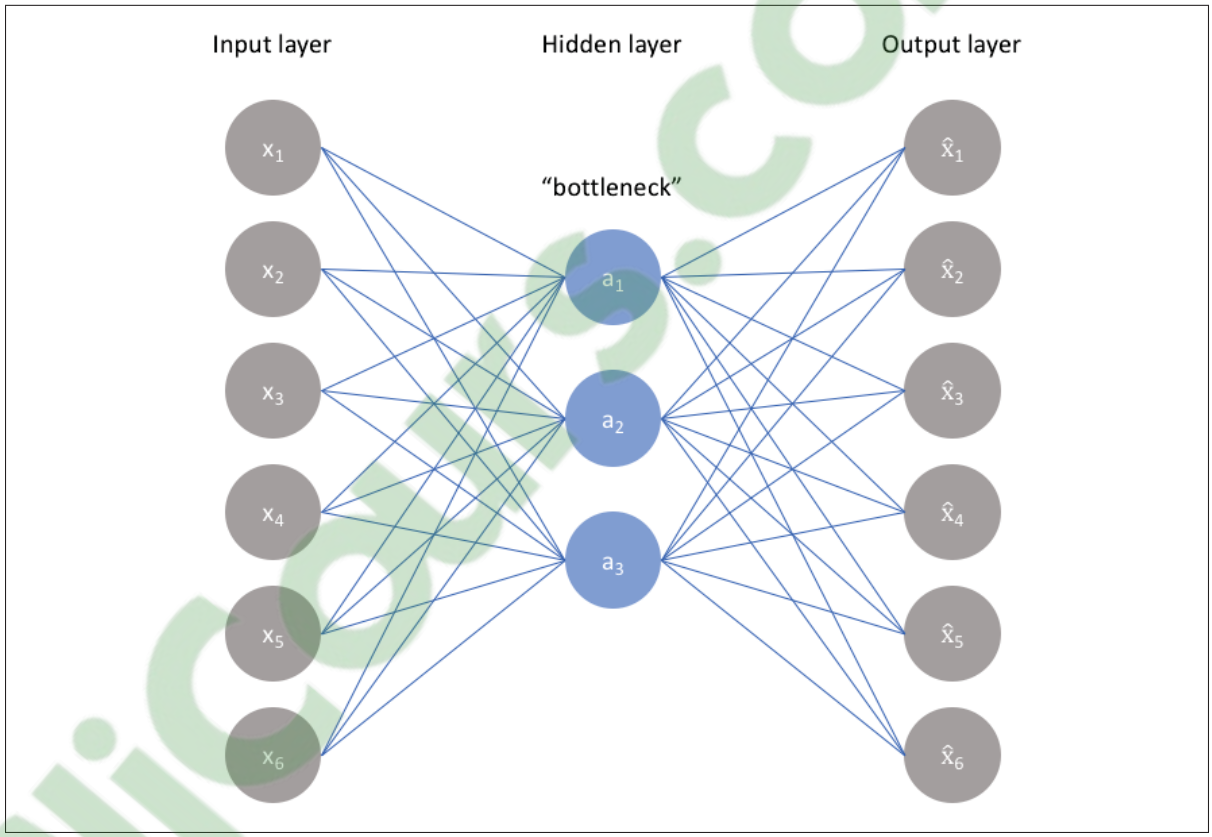

Figure 1.5 A simple autoencoder architecture. Source: www.jeremyjordan.me/autoencoders/

Autoencoders are composed of an encoder and a decoder. The encoder transforms an input vector $\boldsymbol{x}$ into its compressed representation $\boldsymbol{c}$, or *code* vector. The decoder then maps the code vector $\boldsymbol{c}$ back to a reconstructed input vector $\hat{\boldsymbol{x}}$. The compressed representation $\boldsymbol{c}$ is obtained by designing a Artificial Neural Network (ANN) which has a bottleneck hidden layer (see figures 1.5 and 1.6 for graphical depictions of the architecture). The intuition behind the ANN autoencoder architecture is that if a compressed representation $\boldsymbol{c}$ allows for a good reconstruction of its input, then it means that it has retained significant amount of information about said input.

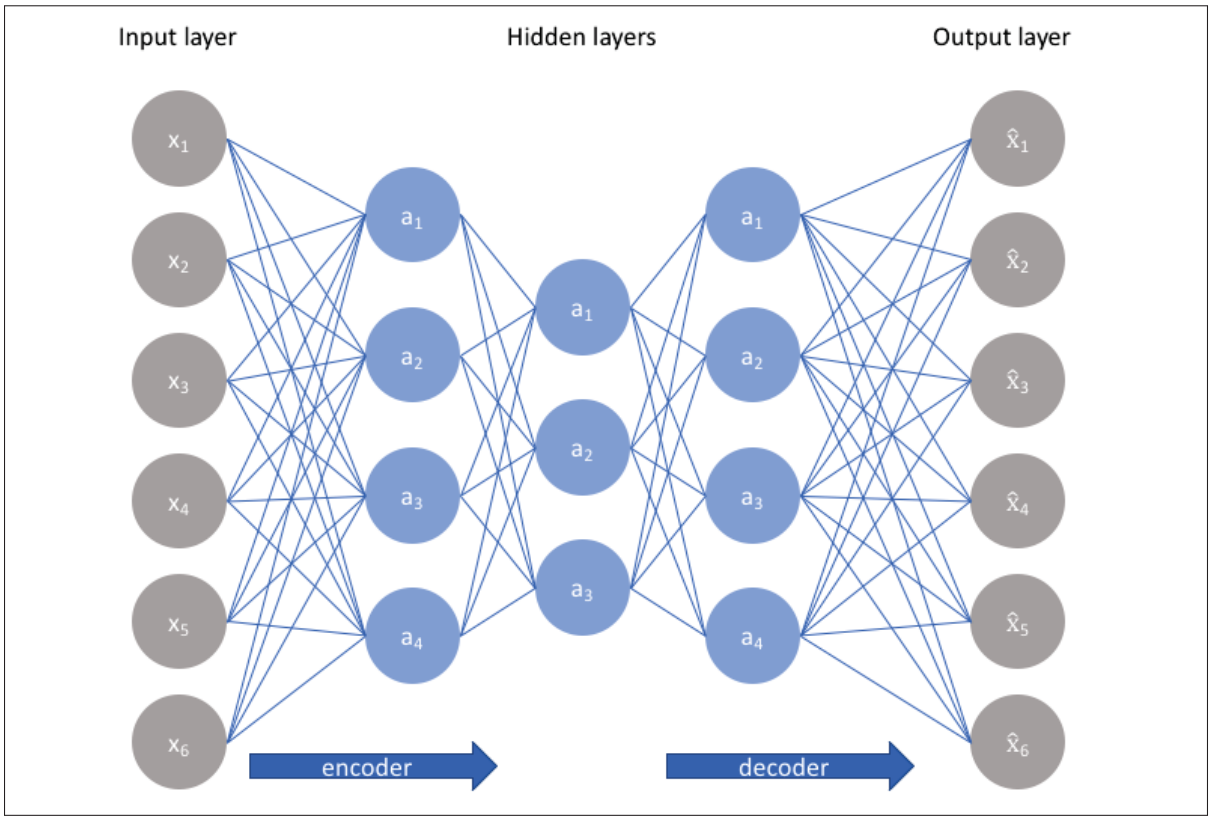

Figure 1.6 A deep autoencoder architecture. Source: www.jeremyjordan.me/autoencoders/

Autoencoders are trained by minimizing the reconstruction error, $J(\mathbb{X}, \hat{\mathbb{X}})$ (see equations 1.5 or 1.6), which measures the difference between the original inputs $\boldsymbol{x} \in \mathbb{X}$ and their corresponding reconstruction $\hat{\boldsymbol{x}} \in \hat{\mathbb{X}}$.

1.2.4 Encoder-Decoders

The Encoder-Decoder (Cho, van Merrienboer, Gulcehre, Bahdanau, Bougares, Schwenk & Bengio, 2014a; Sutskever, Vinyals & Le, 2014) is an ANN architecture that was developed to allow the mapping of sequences to sequences. More formally, Encoder-Decoders learn to estimate the conditional probability $p(\boldsymbol{Y}|\boldsymbol{X})$, where $\boldsymbol{X} = \boldsymbol{x}_0, \ldots, \boldsymbol{x}_m$ is an input sequence of length m and $\boldsymbol{Y} = \boldsymbol{y}_0, \ldots, \boldsymbol{y}_n$ is its corresponding output sequence of length n. Speech recognition, translation and summarization can all be seen as examples of sequence-to-sequence tasks.

The idea behind Encoder-Decoder architectures is to use an RNN encoder to read the input sequence $\boldsymbol{X}$, one item at the time, until a fixed-dimensional vector representation $\boldsymbol{c} = \boldsymbol{h}_m^E$ is obtained, and to then use an RNN decoder to extract the output sequence $\boldsymbol{Y}$ from that vector. The decoder computes each $\boldsymbol{y}_i$ element from $\boldsymbol{Y}$ as follow:

$$\boldsymbol{y}_i = f_D(\boldsymbol{h}_i^D, \boldsymbol{y}_{i-1}, \boldsymbol{c}) \tag{1.13}$$

where:

- $\boldsymbol{y}_i$: decoder output at time step i,
- f_D: function implemented by the RNN Decoder,
- $\boldsymbol{h}_i^D$: decoder's hidden state at time step i,
- $\boldsymbol{c} = \boldsymbol{h}_m^E$: Encoder's hidden state after processing $\boldsymbol{X}$.

Hence, the conditional probability of $\boldsymbol{Y}$ given $\boldsymbol{X}$ can be computed as follow:

$$p(\boldsymbol{Y}|\boldsymbol{X}) = \prod_{i=0}^{n} p(\boldsymbol{y}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}, \boldsymbol{c}) \tag{1.14}$$

In many NLP tasks, each $p(\boldsymbol{y}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}, \boldsymbol{c})$ distribution in equation 1.14 is represented by a softmax[3] over the words of a predefined vocabulary, and sequences are generated by sampling a word from said distribution at each time step.

Encoder-Decoders are typically trained from end to end using the Cross-Entropy loss function (equation 1.6). In practice, training requires that each sequence is appended with an end-of-sequence symbol (e.g. <EOS>) to allow the model to learn a probability distribution over sequences of variable lengths.

[3] The softmax function takes as input a vector of k real numbers and normalizes it so each of its component is in the (0, 1) interval and all components add up to 1.

Over time, different implementation of Encoder-Decoders have emerged, for example based on Convolutional Neural Networks (CNN) (Gehring, Auli, Grangier, Yarats & Dauphin, 2017; Kaiser & Bengio, 2016; Kalchbrenner, Espeholt, Simonyan, van den Oord, Graves & Kavukcuoglu, 2016), which this work do not cover, and the Transformer architecture (Vaswani *et al.*, 2017), which will be seen in detail in section 1.2.6.

1.2.5 Attention Mechanisms

One of the main limitations of basic RNN-based Encoder-Decoders is that the encoder compress all the relevant information of an input sequence to a fixed-dimensional vector representation. This architectural bottleneck was found to hurt performances, especially as the length of sequences starts to grow and gets longer than sequences observed during training (Cho, van Merriënboer, Bahdanau & Bengio, 2014b).

Attention mechanisms (Bahdanau *et al.*, 2014) propose to overcome this limitation by encoding an input sequence into a sequence of vectors and by adaptively choosing a subset of these vectors during decoding in order to better model the task at hand. In other words, attention allows a model to focus on the relevant parts of an input sequence depending on the current output needs. Figure 1.7 offers a visualization of an attention mechanism's decision making process in the context of automatic translation.

Figure 1.7 demonstrates how a particular model paid attention to the correct parts of the French input sequence when it correctly translated *zone économique européenne* to *European Economic Area*. In French, the order of these words is reversed (*Area Economic European*). Every other word in the sentence have the same order in both French and its English translation.

When using Attention mechanisms, the conditional probability from equation 1.14 is redefined as follow:

$$p(\boldsymbol{y}_i|\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}, \boldsymbol{c}_i) = f_A(\boldsymbol{h}_i^D, \boldsymbol{y}_{i-1}, \boldsymbol{c}_i) \tag{1.15}$$

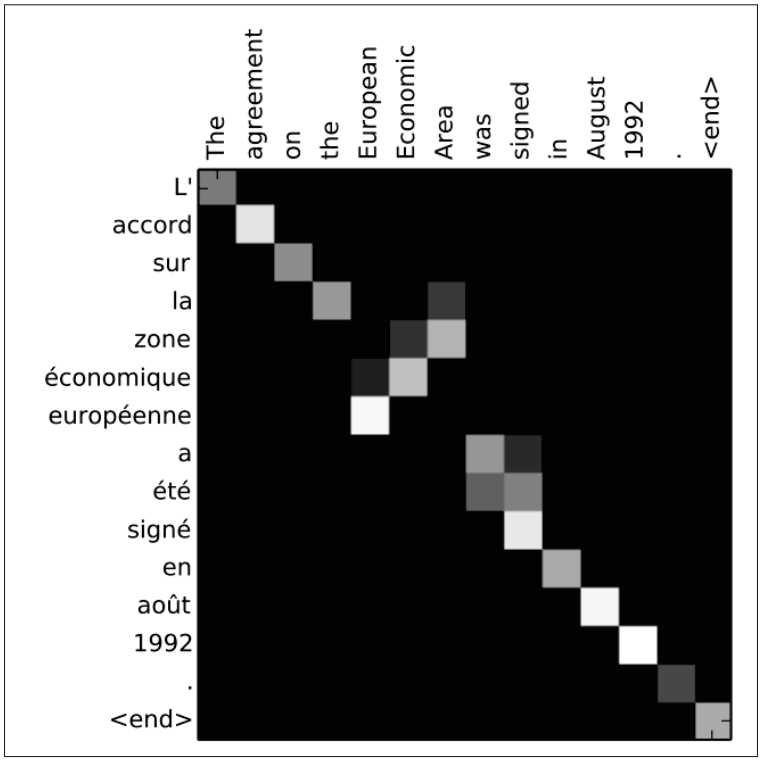

Figure 1.7 Visualization of an attention mechanism's decision making process in the context of automatic translation. In this representation, a light shade indicates a high level of attention and a darker shade indicates a low level of attention. Source: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

One important difference between basic Encoder-Decoders and Encoder-Decoders using attention mechanisms is that when using an attention mechanism the probability is conditioned on a distinct code vector $\boldsymbol{c}_i$ for each target element $\boldsymbol{y}_i$. The code vector $\boldsymbol{c}_i$ is computed using a sequence of hidden states $\boldsymbol{h}_0^E, \ldots, \boldsymbol{h}_m^E$ mapped from an input sentence $\boldsymbol{X} = \boldsymbol{x}_0, \ldots, \boldsymbol{x}_m$ by a bidirectional RNN[4] encoder. One important characteristic of these hidden states sequences is that each $\boldsymbol{h}_i^E$ hidden state contains information about the whole input sequence, but has a

[4] Bidirectional RNNs connect two hidden layers of opposite directions to the same output.

strong focus on the elements surrounding the i-th item of the input sequence. Code vectors $\boldsymbol{c}_i$ are computed as a weighted sum of $\boldsymbol{h}^E$ hidden states:

$$\boldsymbol{c}_i = \sum_{j=0}^{m} \alpha_{ij} \boldsymbol{h}_j^E \tag{1.16}$$

where the weight α_{ij} of each hidden state $\boldsymbol{h}_j^E$ is computed as follow:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=0}^{m} \exp(e_{ik})} \tag{1.17}$$

and

$$e_{ij} = a(\boldsymbol{h}_{i-1}^D, \boldsymbol{h}_j^E) \tag{1.18}$$

is an alignment model which scores how well input elements around position j match the output element at position i. This alignment model is implemented using a Feed-Forward Neural Network which is jointly trained with all the other components of the Encoder-Decoder using the same lost function. Figure 1.8 offers a visualization of the computation of attention scores.

Over time, different types of attention mechanisms have emerged. Notable examples include multi-dimentional attention (Wang, Pan, Dahlmeier & Xiao, 2017), hierarchical attention (Ji, Wang, Toutanova, Gong, Truong & Gao, 2017; Yang, Yang, Dyer, He, Smola & Hovy, 2016) and self-attention mechanisms (Vaswani *et al.*, 2017). Self-attention mechanisms, also called scaled dot-product attention, is seen in detail in section 1.2.6.

1.2.6 Transformer

One of the main limitations of RNNs is that their inherent sequential nature prohibits parallelization within training examples. The issue becomes more apparent as input sequences get longer

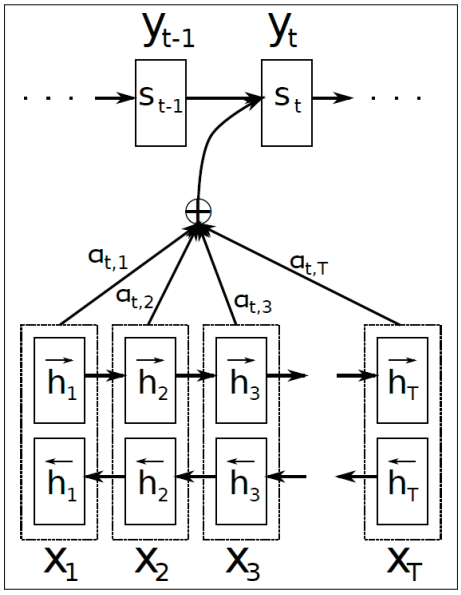

Figure 1.8 Visualization of the computation of attention scores. Source: Bahdanau *et al.* (2014)

because memory constraints limit the size of example batches. The Transformer (Vaswani *et al.*, 2017) is an Encoder-Decoder architecture designed to allow significantly more parallelization than its RNN-based counterparts. It does so by avoiding the use of RNNs in favor of attention mechanisms and Feed-Forward Neural Networks. Figure 1.9 offers a graphical depiction of the Transformer architecture.

The encoder component of a Transformer is of a stack of n encoder layers, where each layer has an identical structure which consists of an attention sub-layer and a feed-forward sub-layer. A residual connection (He, Zhang, Ren & Sun, 2016) is employed around each of the two sub-layers, followed by layer normalization (Ba, Kiros & Hinton, 2018). More formally, the final output of each sub-layer is LayerNorm($\boldsymbol{X}$ + Sublayer($\boldsymbol{X}$)), where Sublayer($\boldsymbol{X}$) is the function implemented by the sub-layer itself.

Transformers employ a variant of attention mechanisms called scaled dot-product attention. This type of attention mechanism uses query and key matrices $\boldsymbol{Q}$ and $\boldsymbol{K}$ of dimension d_k as inputs as

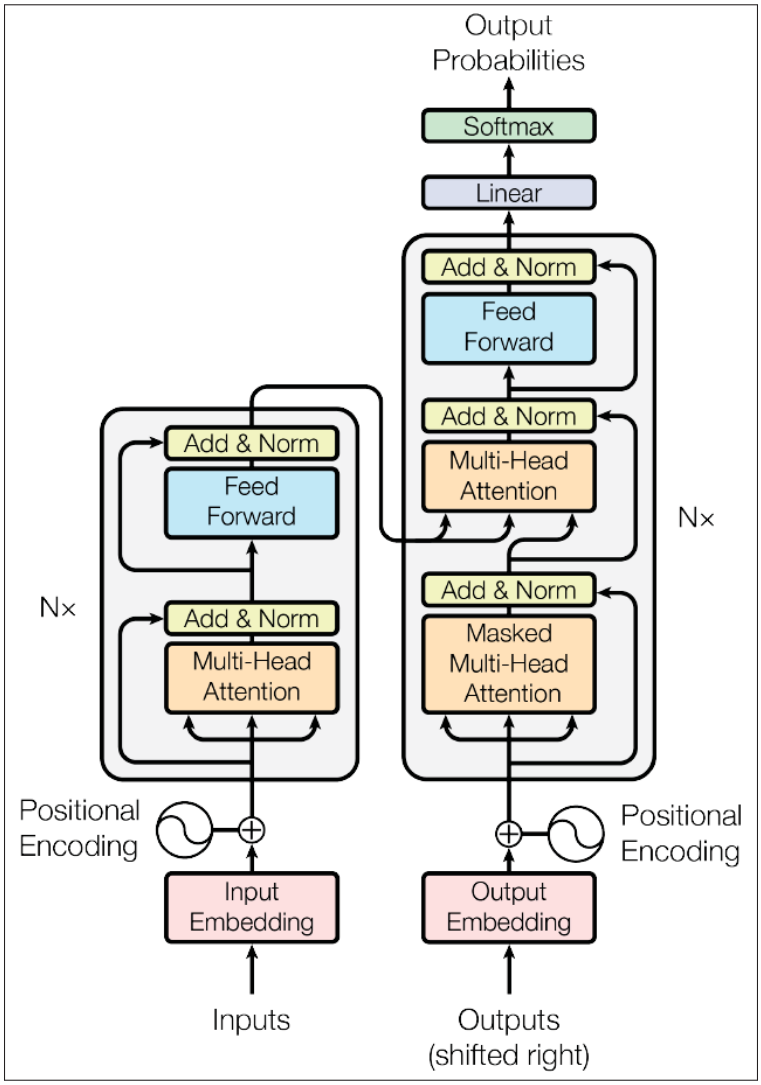

Figure 1.9 Visualization of the Transformer architecture. Source: Vaswani *et al.* (2017)

well as value matrix $\boldsymbol{V}$ of dimension d_v. Attention scores are obtained by first computing the dot products of queries and keys and then scaling down the result by dividing by $\sqrt{d_k}$. A softmax function is finally applied to obtain the weights on the values:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{1.19}$$

In practice, Transformers do not compute attention with d_{model}-dimensional query, key and value matrices but h different learned linear projections of d_k, d_k and d_v dimensions instead. The attention function is applied in parallel for each projected versions of query, key and value matrices, yielding h different d_v-dimensional intermediary output value matrices. Final values are obtained by concatenating the intermediary values and projecting once again. This process is called multi-head attention and is computed as follow:

$$\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\boldsymbol{W}^O \tag{1.20}$$

where $\text{head}_i = \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V)$ and the projections are parameter matrices $\boldsymbol{Q}\boldsymbol{W}_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $\boldsymbol{K}\boldsymbol{W}_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $\boldsymbol{V}\boldsymbol{W}_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $\boldsymbol{W}^O \in \mathbb{R}^{hd_v \times d_{model}}$. Multi-head attention is an important part of the Transformer architecture as it allows to jointly attend to information from different representation sub-spaces at different positions. Figure 1.10 offers a visualization of multi-head attention.

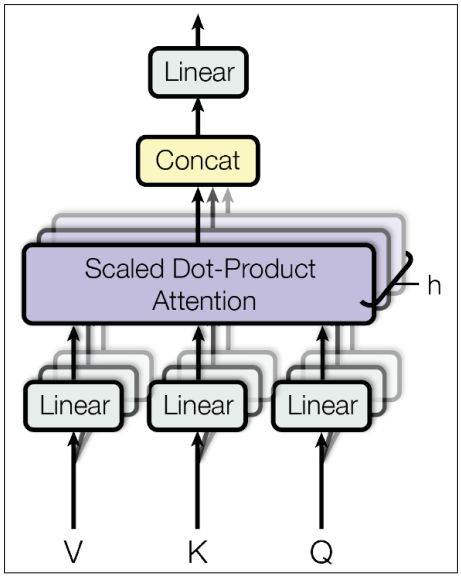

Figure 1.10 Visualization of the multi-head attention mechanism. Source: Vaswani *et al.* (2017)

Similar to its encoder component, the Transformer decoder consists of a stack of n identical decoder layers. However, decoder layers are composed of three different sub-layers: the same two sub-layers as an encoder layer and an additional multi-head attention sub-layer to compute attention scores over the output of the encoder. Furthermore, the first attention sub-layer in a decoder stack is modified to prevent from attending to subsequent (future) positions. This masking used in combination of an offset of one position applied to the output ensures that the predictions for position i can depend only on the known outputs at positions before i.

Both the encoder and decoder Feed-Forward sub-layers are two layers deep with a ReLU (equation 1.4) activation in between:

$$\text{FFNN}(\boldsymbol{X}) = \text{ReLU}(\boldsymbol{W}_1^T \boldsymbol{X})^T \boldsymbol{W}_2 \tag{1.21}$$

Since the Transformer does not use recurrence, information about position of elements in a sequence is injected with the use of Positional Encodings (PE) at the bottoms of the encoder and decoder components. PEs are computed using sinusoidal functions:

$$\text{PE}_{(p,2i)} = \sin\left(\frac{p}{10000^{\frac{2i}{d}}}\right) \tag{1.22}$$

$$\text{PE}_{(p,2i+1)} = \cos\left(\frac{p}{10000^{\frac{2i}{d}}}\right) \tag{1.23}$$

where p is the position of an element in a sequence and $d = d_{model}$ the dimensionality of the input. Equation 1.22 is used for even dimensions and equation 1.23 for odd dimensions. Figure 1.11 offers a visualization of Transformers' Positional Encodings.

1.2.7 Encoder-Decoder Output Control

Output control of Encoder-Decoders is a research topic that recently started to gain more attention in the field of abstractive summariation (Fan, Grangier & Auli, 2018; Kikuchi *et al.*, 2016; Liu, Luo & Zhu, 2018; Takase & Okazaki, 2019). The motivation behind the development of this

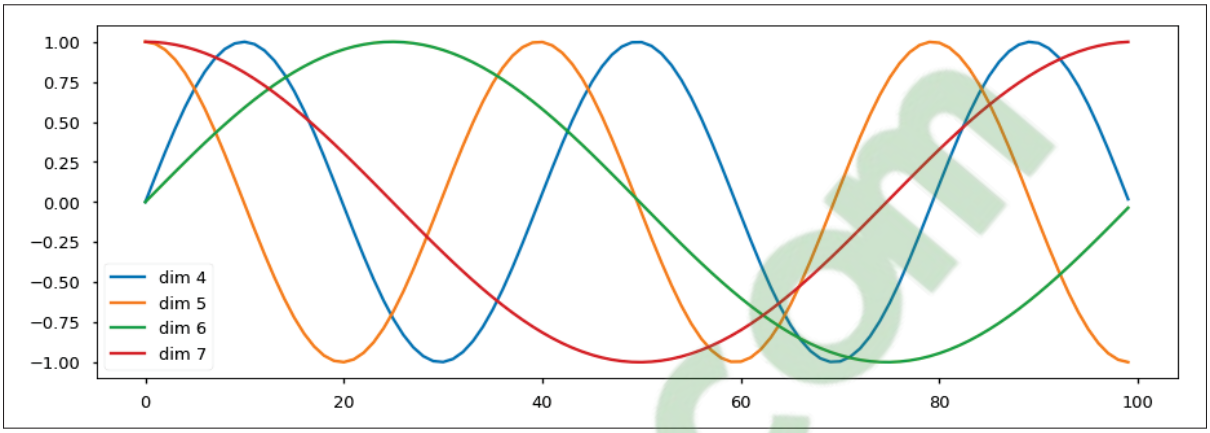

Figure 1.11 Visualization of Positional Encodings. Source: https://nlp.seas.harvard.edu/2018/04/03/attention.html

capability is to allow to shape a model's output to a particular length or style. One of the first attempt is from (Kikuchi *et al.*, 2016), where four different methods of output length control for LSTM-based Encoder-Decoders were investigated. Out of these four methods, two were shown to perform better than the others: LenEmb and LenInit.

With the LenEmb method, the model's decoder is modified so that the number of tokens[5] left to generate in order to obtain an output sequence of the desired length can be passed as input at each decoding step. This new input is encoded as an $\boldsymbol{e}_2(l_t) \in \mathbb{R}^d$ embedding, which is then parameterized by a $\boldsymbol{W}_{le} \in \mathbb{R}^{d \times n}$ matrix of learned parameters where d is the size of the decoder's hidden states and n is the number of potential desired length types. Figure 1.12 illustrates the LenEmb modified decoder.

The remaining length is initialized after the encoding process is completed and updated during the decoding process as follows:

[5] Tokens are the results of a process in which a long sequence of characters is separated into a sequence of small groups of characters following a prespecified algorithm (Manning, Raghavan & Schütze, 2008). Examples of tokens include words, parts of words, integers, reals, dates, punctuation characters and other symbols.

$$l_0 = t \tag{1.24}$$

$$l_{i+1} = \begin{cases} 0 & l_i - \text{token}(\boldsymbol{y}_i) \leq 0 \\ l_i - \text{token}(\boldsymbol{y}_i) & \text{otherwise} \end{cases} \tag{1.25}$$

where:

- t: target output length,
- l_i: remaining length to generate at decoding step i,
- $\boldsymbol{y}_i$: generated sequence at decoding step i,
- token($\boldsymbol{y}_i$): number of tokens in $\boldsymbol{y}_i$.

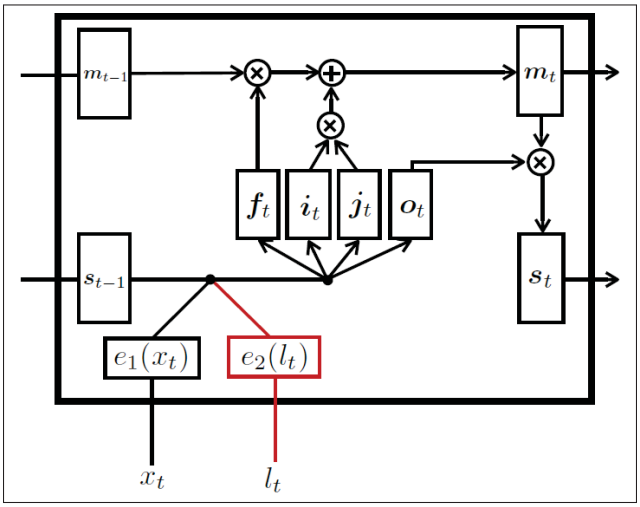

Figure 1.12 LenEmb: using the remaining length to generate as an additional input to the LSTM decoder. Source: (Kikuchi *et al.*, 2016)

Finally, the LenInit method inputs the desired output length once while initializing the decoder's hidden state as follows:

$$\boldsymbol{s}_0 = \overleftarrow{\boldsymbol{h}}_1 \tag{1.26}$$

$$\boldsymbol{m}_0 = t\boldsymbol{b}_c \tag{1.27}$$

where:

- $\boldsymbol{s}_0$: decoder initial hidden state,
- $\overleftarrow{\boldsymbol{h}}_1$: encoder's last backward hidden state,
- $\boldsymbol{m}_0$: decoder initial memory cell state,
- $\boldsymbol{b}_c \in \mathbb{R}^h$: trainable parameter,
- t: target output length.

Figure 1.13 illustrates the LenInit modified decoder.

Takase & Okazaki (2019) propose an output length control mechanism for Transformer-based Encoder-Decoder. The proposal consists of two extensions to standard Transformer Positional Encodings (PE) so they can represent distance from a target output length: Length-Difference Positional Encodings (LDPE) and Length-Ratio Positional Encodings (LRPE). These extensions replace equations 1.22 and 1.23 by equations 1.28 and 1.29 (or 1.30 and 1.31) on the decoder side. LDPE and LRPE are defined as follow:

$$\text{LDPE}_{(p,2i)} = \sin\left(\frac{t-p}{10000^{\frac{2i}{d}}}\right) \tag{1.28}$$

$$\text{LDPE}_{(p,2i+1)} = \cos\left(\frac{t-p}{10000^{\frac{2i}{d}}}\right) \tag{1.29}$$

$$\text{LRPE}_{(p,2i)} = \sin\left(\frac{p}{t^{\frac{2i}{d}}}\right) \tag{1.30}$$

$$\text{LRPE}_{(p,2i+1)} = \cos\left(\frac{p}{t^{\frac{2i}{d}}}\right) \tag{1.31}$$

where t represents the target length and p and d are as per equations 1.22 and 1.23. Incorporating standard Transformer PE to LDPE or LRPE by summing the embeddings (LDPE+PE or LRPE+PE) also provide viable options.

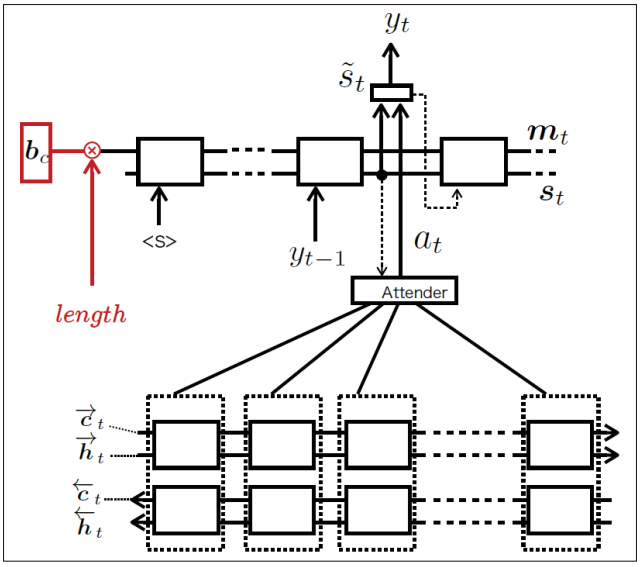

Figure 1.13 LenInit: output length managed via initialization of the decoder's memory cell state. Source: (Kikuchi *et al.*, 2016)

During training, the correct output value is assigned to the t variable. During testing, output length can be controlled by setting t to the desired length.

1.2.8 Artificial Neural Networks and Natural Languages

Artificial Neural Networks require their input to be vectors of continuous values. For this reason, textual data must be transformed before it can be submitted to an ANN. NLP practitioners typically use a set of techniques called word embeddings to perform this transformation.

The simplest technique to transform words into vectors is probably one-hot encoding. This type of encoding maps words from a vocabulary of size k to vectors of k bits, with all legal combinations of bits being restricted to those with a single high (1) bit while every other bit

is set to low (0). For example, using this technique a vocabulary $\mathbb{V} = \{\text{red}, \text{green}, \text{blue}\}$ could be mapped as follows: red = $[0, 0, 1]$, green = $[0, 1, 0]$ and blue = $[1, 0, 0]$. In practice, simple approaches such as one-hot encoding are problematic because they tend to produce very large and sparsely used vector spaces as the vocabulary size increases. Furthermore, they often fail to take any semantic or contextual information into account. More complex word embedding approaches aim to alleviate these issues.

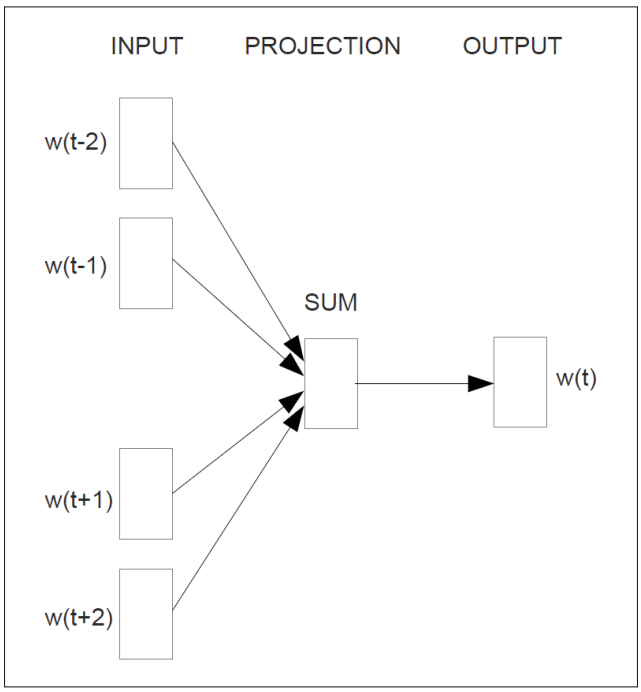

Figure 1.14 Visualization of the usage of a window of words spanning from token $w(t-2)$ to $w(t+2)$ in the process of computing the $w(t)$ WORD2VEC embedding. Source: Mikolov *et al.* (2013)

Among the most popular types of word embeddings are WORD2VEC (Mikolov *et al.*, 2013) and GLOVE (Pennington, Socher & Manning, 2014). Embeddings generated using the WORD2VEC technique are produced by first training a FFNN on the Language Modeling (LM) task in order to find efficient representations of words. Once the FFNN is trained, the weights of its hidden layer, usually called the embedding matrix, are reused in order to map one-hot word encodings to vectors of a smaller dimensionality.

Embeddings generated using the GloVe technique on the other hand are produced by constructing a large *word* × *word* cooccurrence matrix where each intersection counts how frequently a word (row) is found in some context (columns) in a large corpus. One important aspect of the both approaches is that they make use of *context* in order to encode semantic information into the final representation.

Figure 1.14 offers a visualization of the usage of a window of words in order to represent context in the process of computing a word2vec embedding. Figure 1.15 gives a visual representation of how semantic information can be encoded into word embeddings.

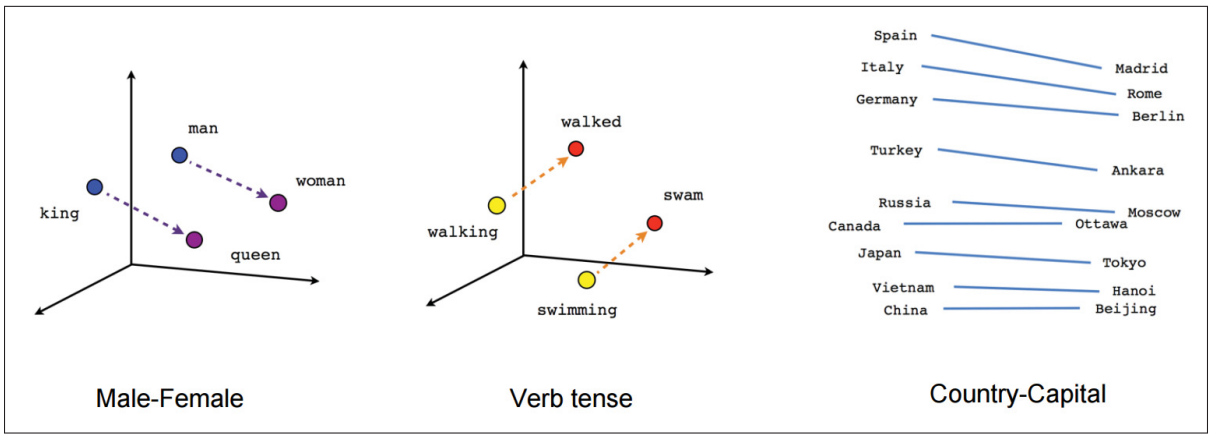

Figure 1.15 Visualization of how word embeddings encode semantic information into a vector space. Source: https://medium.com/deeper-learning/glossary-of-deep-learning-word-embedding-f90c3cec34ca

Many new approaches for the mapping of words to vectors have started to appear in recent years (Howard & Ruder, 2018; Peters, Neumann, Iyyer, Gardner, Clark, Lee & Zettlemoyer, 2018), one of the most influential ones being *Bidirectional Encoder Representations from Transformers*, or Bert for short (Devlin, Chang, Lee & Toutanova, 2018).

Bert provides a pretrained language representation model that extend the idea of word embeddings to include further contextual information. It is trained on the *masked language modeling* and *next sentence prediction* tasks using a large-scale dataset of paired sentences called *sequences*.

The masked language modelling task consists of randomly masking some tokens from the input and asking a model to predict what token is under each particular mask based only on its surrounding context.

The next sentence prediction task consists of selecting pairs of sentences from a monolingual corpus where sentence B follows sentence A 50% of the time. The model is then asked to predict if sentence B follows sentence A, or not.

Bert's training dataset is preprocessed by inserting two special tokens into the sequences: [CLS] and [SEP]. The [CLS] token is inserted at the start of each sequence so its output representation can be used to aggregate information for classification tasks. The second special token, [SEP], is inserted between sentences of a sequence to clearly indicate their boundaries. A preprocessed Bert sequence is represented as a sequence of input embeddings $\boldsymbol{X} = \boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_m$ where each $\boldsymbol{x}_i$ is the sum of three different embeddings: a token embedding, a segment embedding and a position embedding. Token embeddings are provided by a WordPiece tokenizer (Wu *et al.*, 2016) using a vocabulary of 30,000 tokens. Segment embeddings are two learned embeddings designed to help the model differentiate sentences of a sequence (sentence A is assigned an $\boldsymbol{e}_A$ embedding and sentence B an $\boldsymbol{e}_B$ embedding). Finally, position embedding are standard Transformer Positional Encodings (equations 1.22 and 1.23) set to support sequences of up to 512 tokens. Section 1.2.6 describes Transformer Positional Encodings in detail.

Preprocessed Bert sequences are submitted to a deep biderectionnal Transformer Encoder (Vaswani *et al.*, 2017) for training on the aforementioned tasks. Once training is completed, representations provided by the resulting Encoder can then typically be fine-tuned on task-specific objectives by adding one or more output layer(s).

1.3 Previous work

1.3.1 Supervised Abstractive Summarization

Supervised approaches have recently made great progress in the field of abstractive single-document summarization. Rush *et al.* (2015) first introduced the usage of RNN-based Encoder-Decoders and attention mechanisms for the task of sentence compression. Nallapati *et al.* (2016) soon followed and introduced the CNN/DailyMail corpus, the first large-scale dataset for the training of abstractive summarizers. Then, See *et al.* (2016) proposed two important improvements to the basic architecture: a pointer-generator mechanism to help summarizers manage out-of-vocabulary tokens, and a coverage mechanism to reduce the amount of unnecessary repetitions that plagued the outputs generated by this first wave of deep neural summarizers. Recent proposals have switched to the more modern Transformer based Encoder-Decoder architecture and explored various form of pretrained representations (Lewis *et al.*, 2019; Liu & Lapata, 2019; Zhang *et al.*, 2019).

Bert for Summarization, or BertSum (Liu & Lapata, 2019), is a document-level encoder specifically designed to provide deep bidirectionnal representations for the task of single-document summarization (extractive and abstractive). One key difference between Bert and BertSum lies in the preprocessing of the input data.

In order to leverage the pretrained Bert model for summarization purposes, BertSum's preprocessing procedure considers an input document as a sequence of n sentences. Bert's special tokens [CLS] and [SEP] are inserted before and after each sentence of a sequence. Segment embeddings are used at interval depending on whether $sent_i$ is odd or even (document $\boldsymbol{D} = [sent_1, sent_2, sent_3, sent_4, sent_5]$ would have the $[\boldsymbol{e}_A, \boldsymbol{e}_B, \boldsymbol{e}_A, \boldsymbol{e}_B, \boldsymbol{e}_A]$ embeddings assigned to it). Finally, BertSum uses Transformer Position Encodings that can be fine-tuned to support sequences longer than 512 tokens. See figure 1.16 for a comparison of Bert's and BertSum's encoders.

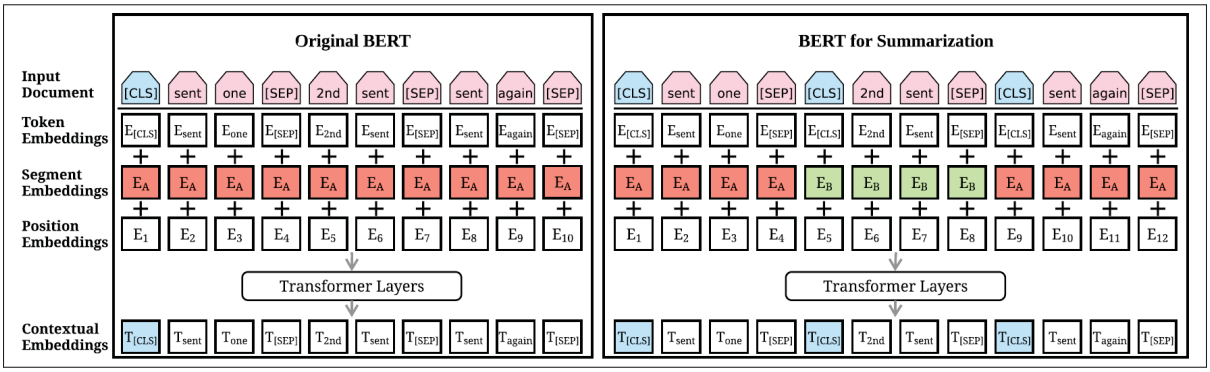

Figure 1.16 Comparison of BERT's and BERTSUM's encoders.
Source: Liu & Lapata (2019)

BERTSUMEXTABS (Liu & Lapata, 2019) is an abstractive summarization model built on top of the BERTSUM encoder. The encoder is first fine-tuned on the extractive summarization task before it is combined with a randomly-initialised Transformer decoder (Vaswani *et al.*, 2017) for further fine-tuning on the abstractive summarization task. This two-stage training procedure was found to increase performance of the final abstractive model. Two different optimizers are used to account for the fact that the encoder is pretrained while the decoder is not. Training is conducted using a standard Cross-Entropy loss (equation 1.6) for the prediction of $\hat{y}_i$ tokens against y_i reference tokens.

1.3.2 Unsupervised Abstractive Summarization

Research in unsupervised single-document summarization historically focused on extractive approaches (Erkan & Radev, 2004; Hirao, Yoshida, Nishino, Yasuda & Nagata, 2013; Marcu, 1997; Mihalcea, 2004; Parveen, Ramsl & Strube, 2015; Yin & Pei, 2015). Recent work in this area include (Fevry & Phang, 2018), in which a denoising autoencoder (Vincent, Larochelle, Bengio & Manzagol, 2008) is trained to remove and reorder words from an input sequence and (Zheng & Lapata, 2019), which uses BERT (Devlin *et al.*, 2018) to capture sentential meaning and compute sentence similarity for the purpose of sentence selection.

Most attempts at unsupervised abstractive summarization so far have focused on the task of sentence compression, which have been described as a scaled-down version of the summarization problem (Knight & Marcu, 2002). Zhou & Rush (2019) for example propose the use of two language models and a product-of-experts criteria to learn to compress sentences in an unsupervised fashion. BottleSum (West, Holtzman, Buys & Choi, 2019) is another example where a pretrained language model is fine-tuned on the abstractive sentence compression task using summaries from an extractive summarizer trained in an unsupervised fashion. Similar to this work, Schumann (2018) proposes the use of a RNN-based variational autoencoder (Kingma & Welling, 2014) and the LenEmb output length control mechanism (Kikuchi *et al.*, 2016) for unsupervised sentence compression.

Unsupervised abstractive summarization of news have only recently started to gain attention. For example, Wang & Lee (2018) propose an autoencoder based on Generative Adversarial Networks (GAN) (Goodfellow, Pouget-Abadie, Mirza, Xu, Warde-Farley, Ozair, Courville & Bengio, 2014) trained to summarize news articles using large datasets of unpaired documents and summaries. Unfortunately, the necessity to provide summaries during training seems to render this approach incompatible with most unsupervised learning use cases. Furthermore, the method did not perform well when transfer learning was involved.

More recently, progress has been made by training Language Models (LM) on large amounts of unlabeled text using models based on the Transformer architecture (Vaswani *et al.*, 2017). Even though these attempts do not necessitate an aligned corpus, they still rely on the usage of a training corpus containing summaries of some sort. Gpt-2 (Radford, Wu, Child, Luan, Amodei & Sutskever, 2019) for example learns to generate summaries by identifying "TL;DR" (Too Long; Didn't Read) tokens in the WebText corpus[6]. To induce a summarization behavior, a TL;DR token is appended to a long document fed as input and the LM is then prompted to predict the text that follows. The Language Models trained using Gpt-2 perform remarkably well on numerous NLP tasks, but abstractive summarization is not one of them.

[6] WebText is a corpus scraped from outbound links from Reddit, a social media platform.

Finally, SummAE (Liu, Chung & Ren, 2019a) uses a denoising autoencoder and self-supervised pretraining to learn to summarize paragraph-length short stories in an unsupervised fashion. The model is trained to encodes sentence-length and paragraph-length documents to a shared latent vector representation space. The decoder input is prepended with a special token to signal whether the output should be of sentence or paragraph length. Once training is completed, summarization is achieved by decoding a sentence-length sequence from an encoded paragraph-length document. As with all previous unsupervised abstractive summarization proposals, the summaries generated by SummAE are not of an high enough quality to consider the model for industrial applications.

1.3.2.1 MeanSum: unsupervised multi-document abstractive summarization

MeanSum (Chu & Liu, 2018) is one of the first efforts to provide an unsupervised abstractive multi-document model for generating generic summaries. It is trained on the publicly available Yelp[7] corpus of reviews.

The two most important components of the model are an autoencoder and a summarizer. The autoencoder's function is to generate representations of each input reviews and constrain the generated summaries to a specified language domain (vocabulary). The summarizer's function is to generate summaries that are semantically similar to the input documents.

Both components are mLSTM encoder-decoders (Krause, Lu, Murray & Renals, 2016). Furthermore, the two encoders' weights are tied and the two decoders' weights are tied. Both encoder-decoders weights are initialized using the weights from the same pre-trained language model. The overall architecture is shown in Figure 1.17.

Suppose an invertible tokenizer T that maps text documents from a corpus $\mathbb{D}$ to a set of token sequences $T(\mathbb{D})$ using a fixed vocabulary. Furthermore, let $\mathbb{V}$ be a subset of tokenized documents with a maximum length of m so that $\mathbb{V} \subset T(\mathbb{D})$. Given a set of k tokenized reviews of a business, $\{x_1, x_2, ..., x_k\} \in \mathbb{V}$, MeanSum produces a tokenized summary s so that $s \in T(\mathbb{D})$.

[7] https://www.yelp.com/dataset/challenge

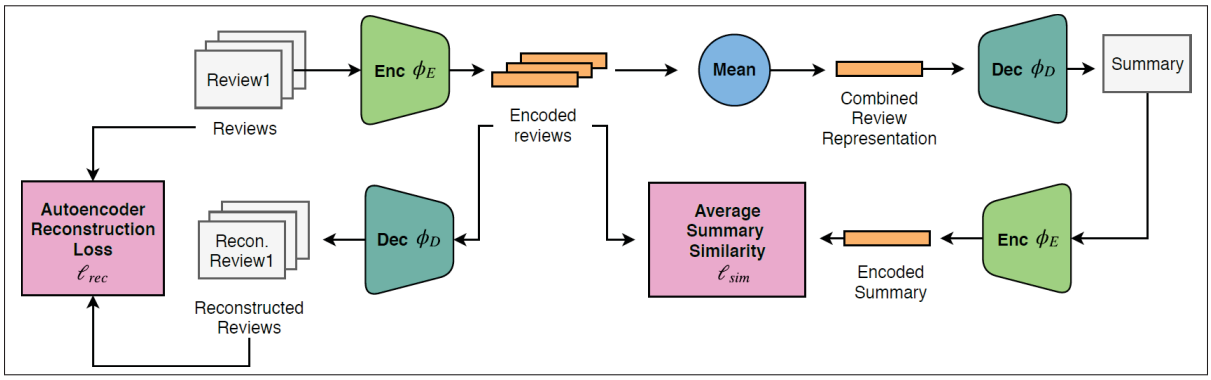

Figure 1.17 The MeanSum model architecture. Source: Chu & Liu (2018)

MeanSum's encoder, ϕ_E, maps reviews to real-vector codes ($\phi_E : \mathbb{V} \mapsto \mathbb{R}^n$) so that $\phi_E(x_j) = z_j$, where z_j is the concatenation of the final hidden and cell states of an mLSTM after processing the t tokens from x_j ($z_j = [\boldsymbol{h}_t, \boldsymbol{c}_t]$). MeanSum's decoder, ϕ_D, is a second mLSTM which initializes its hidden and cell states with z_j and defines a distribution over $\mathbb{V}$ conditioned on z_j so that $\phi_D(z_j) = p(x|z_j)$. The autoencoder component is trained to reconstruct the original reviews using teacher-forcing (Williams & Zipser, 1989) and a standard cross-entropy loss:

$$J_{REC}(\{x_1, x_2, ..., x_k\}, \phi_E, \phi_D) = \sum_{j=1}^{k} J_{CE}(x_j, \phi_D(\phi_E(x_j))) \tag{1.32}$$

where:

- J_{REC}: recreation loss
- x_j: tokenized review of a business
- ϕ_E: encoder
- ϕ_D: decoder
- J_{CE}: cross-entropy loss function from equation 1.6
- k: number of tokenized reviews

MeanSum's summarization module combines a set of encoded reviews $\{z_1, z_2, \ldots, z_k\}$ by computing the mean over the corresponding hidden and cell states, $\bar{z} = [\bar{h}_t, \bar{c}_t]$, and uses ϕ_D to decodes $\bar{z}$ into the summary s. Using the same ϕ_D decoder as the autoencoder constrains the generated summary to the space of reviews ($s \in \mathbb{V}$). The summary is then re-encoded using $\phi_E(s) = [\boldsymbol{h}_s, \boldsymbol{c}_s]$ so a similarity loss can be computed and the summary can be further constrained to be semantically similar to the original reviews. Average cosine distance is used between the hidden states $\boldsymbol{h}_j$ of each encoded review and the hidden state $\boldsymbol{h}_s$ of the encoded summary:

$$s \sim \phi_D(\bar{z}) \tag{1.33}$$

$$J_{SIM}(\{x_1, x_2, ..., x_k\}, \phi_E, \phi_D) = \frac{1}{k} \sum_{j=1}^{k} d_{cos}(\boldsymbol{h}_j, \boldsymbol{h}_s) \tag{1.34}$$

where:

- J_{SIM}: similarity loss
- x_j: review of a business
- ϕ_E: encoder
- ϕ_D: decoder
- k: number of tokenized reviews
- d_{cos}: average cosine distance
- $\boldsymbol{h}_s$: encoded summary
- $\boldsymbol{h}_j$: encoded x_j review

The loss function used to optimize MeanSum during training is the sum of the recreation loss and the similarity loss:

$$J_{\text{MeanSum}} = J_{REC} + J_{SIM} \tag{1.35}$$

1.4 Summary

Automatic summarization is concerned with compressing document(s) into a concise and fluent summary while preserving the most important information. Approaches to summarization can be splitted according to different axis, such as extractive vs. abstractive and supervised vs. unsupervised.

Extractive algorithms redact summaries by concatenating relevant portions of the input, while abstractive algorithms generate new texts that may use terms that are not present in the input. It has been observed that human written summaries tend to be abstractive.

The current State-Of-The-Art (SOTA) abstractive summarization models are based on a Neural Network architecture named Encoder-Decoder and trained in a supervised fashion. Unfortunately supervised training require large amount of aligned data which is difficult and expansive to produce. Unsupervised abstractive models on the other hand can be trained using cheaply and easily available data, but the summaries they generate are lacking in quality.

One desirable attribute of summarization systems is to allow to control the length of the generated summaries. A few techniques allowing this feature have been proposed in the past in the context of sentence compression, but it is unknown whether they will successfully generalize to the task of news summarization.

From the information contained in this chapter it becomes evident that the capacity to train abstractive summarization models in an unsupervised fashion would be an ideal approach to fulfil the needs of the R&D project at hand, as well as many other tasks across various industries.

The main challenges identified are: (1) to improve the SOTA in the field of unsupervised abstractive summarization and (2) to adapt a technique of output length control from the task of sentence compression to the task of news summarization.

CHAPTER 2

METHODOLOGY

This chapter describes the datasets employed in this work, along the preprocessing steps that were used to transform them.

Two sets of experiments were conducted: first, a set involving RNN-based Encoder-Decoders and then a second set involving Transformers. The experiments based on Transformers were designed to remedy the shortcomings of the RNN-based experiments, and therefore used different training datasets and preprocessing steps. The structure on this chapter reflects this fact.

The methodology and metrics used for the evaluation of models remained the same during all experiments and are also explained in detail.

2.1 Datasets

Up until recently, the de-facto standard for the evaluation of single-document summarization models have been the DUC datasets (2001 to 2004). These datasets contain about 50 sets of documents each, where a set typically contains slightly over 10 news articles about a particular topic. A summary is provided for each individual article. The size of these datasets is not sufficient to train abstractive summarization models in a supervised fashion, which is one of the reasons why research historically focused on unsupervised extractive approaches.

This limitation started fading away in 2015 with the introduction of the first freely available, large scale summarization corpus: CNN/DailyMail (Hermann, Kočiský, Grefenstette, Espeholt, Kay, Suleyman & Blunsom, 2015). This dataset contains about 93,000 articles from the CNN and 220,000 articles from the Daily Mail websites. Both news providers supply their articles with a number of human redacted highlight sentences summarising the key points of the article. These highlight sentences are concatenated in order to provide proxies to human redacted reference summaries.

XSum (Narayan, Cohen & Lapata, 2018) is a freely available summarization dataset that was harvested from the British Broadcasting Corporation (BBC) website. It contains about 226,000 news articles with matching one-sentence summaries. Summaries were redacted with the goal to answer the question "What is this article about?".

Newsroom is another Freely available large scale summarization dataset. It contains 1.3 million articles and summaries written by authors and editors from 38 major news publications. The summaries were redacted by humans and published with the articles as metadata for social media services and search engines page descriptions.

Other often used large scale summarization datasets are the New York Times annotated corpus (Sandhaus, 2008), English Gigaword (Graff & Cieri, 2003) and English Gigaword Fifth Edition (Parker, Graff, Kong, Chen & Maeda, 2011). The main drawback of these datasets is that they are not freely available to the general public.

Table 2.1 summarizes the most widely used aligned datasets in the field of summarization at the moment of writing.

Table 2.1 Most widely used aligned datasets in the field of summarization

Dataset	Nb. documents	Avg. summary size (nb. words)
CNN/DailyMail	312k	60
DUC 2001	600	100
DUC 2002	600	100
DUC 2003	300	10
DUC 2004	1000	10
Gigaword	4.1M	10
Gigaword v5	9.9M	10
Newsroom	1.3M	30
NYT	1.8M	10
Xsum	226k	20

Unsupervised training can be performed using large large amounts of news article such as provided by the Common Crawl News corpus, also known as CC-News (Nagel, 2016). This

corpus is not considered an aligned dataset for the reason that it does not provide summaries for the articles it contains. According to Liu *et al.* (2019b), the CC-News corpus contains 63M English news articles.

2.2 Preprocessing

2.2.1 RNN-based experiments

RNN-based models were trained on the CNN/DailyMail dataset (Hermann *et al.*, 2015). The dataset is split into three subsets: a training set containing about 92% of the documents, a validation set containing about 4% of the documents and a test set also containing about 4% of the documents. These subsets are sampled as per (See *et al.*, 2016). Since experiments are conducted in an unsupervised fashion, the summaries included in the training and validation subsets are not used as targets but instead treated as input documents themselves, hence doubling the size of these particular sets. This have the effect of exposing the model to texts of summary length. Evaluation is conducted using document and summary pairs from the test set in a supervised fashion. This strategy was designed to allow for unsupervised training and validation of models while maintaining the ability to evaluate and compare results. A successful unsupervised training strategy developed using this methodology could then be used to train models on a different corpus, for example a French corpus, but it would then be necessary to either design an unsupervised evaluation methodology or create a French evaluation dataset (no such dataset exists at the moment of writing).

All three subsets are tokenized using the WordPiece tokenizer (Wu *et al.*, 2016) and filtered so that only documents containing $1,000$ tokens or less remain. Filtering out long documents was necessary due to memory limitations of the available hardware.

Table 2.2 contains a comparison of the distributions of documents lengths (in token counts) between the full CNN/DailyMail dataset and its filtered version. In both versions summaries are considered additional documents. Figure 2.1 displays a visualization of the same comparison.

Note that the large concentration of documents containing from 50 to 100 tokens is a consequence of considering summaries as additional documents.

Table 2.2 Distributions of documents lengths in the CNN/DailyMail datasets (in tokens count)

	CNN/DailyMail	**Filtered CNN/DailyMail**
Mean length	462	345
Standard deviation	500	322
Minimal length	7	7
First quartile	59	56
Median	1295	145
Third quartile	784	637
Maximal length	5210	1000

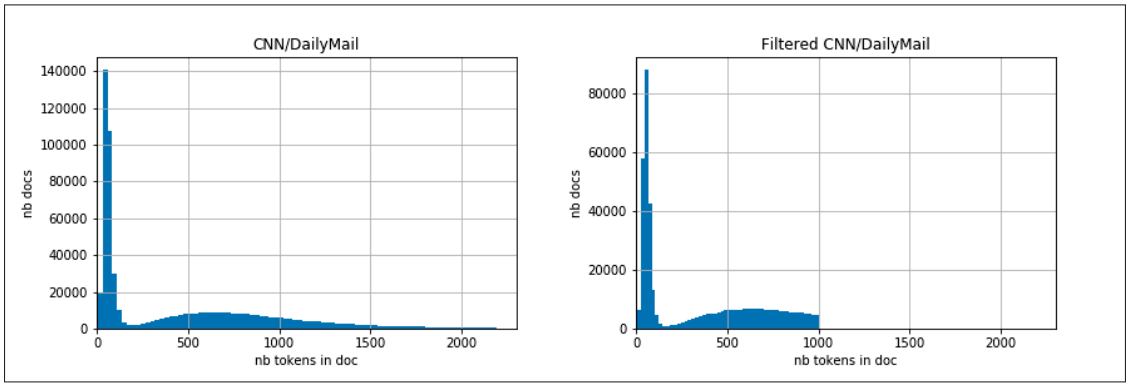

Figure 2.1 Distributions of documents lengths in the CNN/DailyMail datasets (in tokens count)

2.2.2 Transformer-based experiments

As a result from the lessons learned during the RNN-based experiments, the Transformer-based models were trained on the English segment of the CC-News dataset (Nagel, 2016). Documents are collected using `news-please` (Hamborg, Meuschke, Breitinger & Gipp, 2017). Sentences are split using Stanford CoreNLP toolkit (Manning, Surdeanu, Bauer, Finkel, Bethard & McClosky, 2014) and tokenized using WordPiece (Wu *et al.*, 2016) before being submitted to BertSum preprocessing (section 1.3.1).

Two different versions of the CC-News dataset are used. Inspired by (Fan *et al.*, 2018; Liu *et al.*, 2018), the datasets are built by filling discrete bins of documents, each representing a document length range and filled with an approximately equal number of items. The first version of the dataset, denoted CC-News$_{[15;512]}$, contains 5,275,836 documents between 15 and 512 tokens. The second version, denoted CC-News$_{[50;512]}$, contains 5,046,632 documents between 50 and 512 tokens. Documents under 50 tokens were excluded from CC-News$_{[50;512]}$ in an attempt to reduce noise after a manual inspection showed that very short documents often consisted of advertisements or paywall pages. The upper bound of document lengths is fixed to 512 tokens for both versions as this is the maximum length of sequences that can be processed by a standard BERT encoder. Each version of the dataset had 20,000 documents randomly sampled and put aside for validation purposes. Unless specified otherwise, Transformer-based models from chapter 4 are trained and validated using CC-News$_{[50;512]}$.

Table 2.3 summarizes the statistics about the two different versions of CC-News used in this work.

Table 2.3 Statistics about the different versions of CC-News used in this work

Dataset	Nb. documents	Avg. document size (nb. tokens)
CC-News$_{[15;512]}$	5,275,836	240
CC-News$_{[50;512]}$	5,046,632	260

Multiple experiments based on the Transformer architectures involved training as a denoising autoencoder. Three different types of noise were used for these experiments: *Text Infilling*, *Sentence Permutation* and *Extract-N*.

Text Infilling: Inspired by BART (Lewis *et al.*, 2019), this strategy consists of inserting noise by randomly choosing word spans across the source and replacing them with a masking token.

Sentence Permutation: Also inspired by BART, this strategy consists of inserting noise by randomly shuffling source sentences.

Extract-N: Inspired by PEGASUS (Zhang *et al.*, 2019), this strategy consists of inserting noise by setting the target $\boldsymbol{X_T} = \boldsymbol{Y}$ to the firsts n sentences of the source X. The target is contained in the source. Extract-N is based on the lead bias (Kedzie, McKeown & Daumé III, 2018; See *et al.*, 2016; Zhang *et al.*, 2019) which states that according to journalistic conventions, the most important information in a news report usually appears near the beginning of the article.

Finally, *Remove-N* is a novel type of preprocessing strategy designed to generate an aligned summarization dataset from a large, unaligned news corpus ($\mathbb{X} \rightarrow \mathbb{X}_S, \mathbb{X}_T = \mathbb{Y}$). Similar to *Extract-N*, *Remove-N* takes advantage of the lead bias by setting the target $\boldsymbol{X_T} = \boldsymbol{Y}$ to the firsts n sentences of $\boldsymbol{X}$. Contrary to *Extract-N*, the target is removed from the source $\boldsymbol{X_S}$.

2.3 Evaluation

Traditional approaches to evaluation of summaries can be splitted according to two main axis: intrinsic vs. extrinsic and manual (human) vs. automatic (Belz & Reiter, 2006).

Intrinsic evaluations involve rating generated summaries against human written summaries for metrics such as quality, correctness, naturalness and understandability. Extrinsic evaluations focus on evaluating the impact a given summarization system have on a given application, such as measuring the correctness of decisions made in a task based evaluation, or measuring the usage/utility of a given system (Belz & Reiter, 2006).

Historically, summarization systems have been evaluated manually by human subjects (Belz & Reiter, 2006; Mellish & Dale, 1998). However, manual evaluation of summaries on a large scale is costly and cumbersome (Kryściński *et al.*, 2019; Lin, 2004; Over & Yen, 2003). Automatic metrics such as ROUGE (Lin, 2004; Lin & Hovy, 2003) have become the norm over the years because they allow fast and cheap evaluation of models (Kryściński *et al.*, 2019; Owczarzak, Conroy, Dang & Nenkova, 2012).

2.3.1 ROUGE: Recall-Oriented Understudy for Gisting Evaluation

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) was first introduced in (Lin & Hovy, 2003) and further developed in (Lin, 2004). Since its introduction, it has become the standard in automatic evaluation of summaries (Kryściński *et al.*, 2019; Owczarzak *et al.*, 2012).

ROUGE is an intrinsic method of evaluation, as it measures how well a machine generated candidate summary overlaps with human redacted reference(s). It is based on n-gram[1] co-occurrence statistics.

The followings sections give detailed descriptions of two different ROUGE measures: ROUGE-N and ROUGE-L.

2.3.1.1 ROUGE-N: n-gram cooccurrence statistics

ROUGE-N recall is computed as follow:

$$\mathrm{R}_{\mathrm{N}}^{\mathrm{R}} = \frac{\sum_{\boldsymbol{r}\in\mathbb{R}} \sum_{n\text{-}gram_{\boldsymbol{r}}\in\boldsymbol{r}} Count_{match}(n\text{-}gram_{\boldsymbol{r}}, \mathbb{C})}{\sum_{\boldsymbol{r}\in\mathbb{R}} \sum_{n\text{-}gram_{\boldsymbol{r}}\in\boldsymbol{r}} Count(n\text{-}gram_{\boldsymbol{r}}, \boldsymbol{r})} \tag{2.1}$$

where:

- $\mathbb{C}$: sentences from the candidate summary,
- $\mathbb{R}$: sentences from the reference summary(ies) for $\mathbb{C}$,
- $\boldsymbol{r}$: sentence from $\mathbb{R}$,
- $n\text{-}gram_{\boldsymbol{r}}$: n-gram from $\boldsymbol{r}$,
- $Count_{match}(n\text{-}gram_{\boldsymbol{r}}, \mathbb{C})$: maximum number of $n\text{-}gram_{\boldsymbol{r}}$ occurrences in $\mathbb{C}$,

[1] An n-gram is a contiguous sequence of n words from a given sample of text (or speech). An n-gram of size 1, or unigram, corresponds to a single word, while an n-gram of size 2, or bigram, corresponds to a sequence of two words.

- $Count(n\text{-}gram_{\boldsymbol{r}}, \boldsymbol{r})$: number of $n\text{-}gram_{\boldsymbol{r}}$ occurrences in $\boldsymbol{r}$,
- $\mathrm{R}_{\mathrm{N}}^{\mathrm{R}}$: ROUGE-N recall score.

The numerator is the total sum of n-gram matches between a candidate summary $\mathbb{C}$ and reference summary(ies) $\mathbb{R}$, while the denominator is the total sum of n-grams occurrences in $\mathbb{R}$. Therefore, R_N^R is an n-gram recall measure of candidate summary $\mathbb{C}$ with respect to reference summary(ies) $\mathbb{R}$. Accordingly, ROUGE-N precision scores are computed as follow:

$$\mathrm{R}_{\mathrm{N}}^{\mathrm{P}} = \frac{\sum_{\boldsymbol{r} \in \mathbb{R}} \sum_{n\text{-}gram_{\boldsymbol{r}} \in \boldsymbol{r}} Count_{match}(n\text{-}gram_{\boldsymbol{r}}, \mathbb{C})}{\sum_{\boldsymbol{c} \in \mathbb{C}} \sum_{n\text{-}gram_{\boldsymbol{c}} \in \boldsymbol{c}} Count(n\text{-}gram_{\boldsymbol{c}}, \boldsymbol{c})} \tag{2.2}$$

where:

- $\mathbb{C}$: candidate summary,
- $\boldsymbol{c}$: sentence from $\mathbb{C}$,
- $n\text{-}gram_{\boldsymbol{c}}$: n-gram from $\boldsymbol{c}$,
- $\mathbb{R}$: sentences from the reference summary(ies) for $\mathbb{C}$,
- $\boldsymbol{r}$: sentence from $\mathbb{R}$,
- $n\text{-}gram_{\boldsymbol{r}}$: n-gram from $\boldsymbol{r}$,
- $Count_{match}(n\text{-}gram_{\boldsymbol{r}}, \mathbb{C})$: number of $n\text{-}gram_{\boldsymbol{r}}$ occurrences in $\mathbb{C}$,
- $Count(n\text{-}gram_{\boldsymbol{c}}, \boldsymbol{c})$: number of $n\text{-}gram_{\boldsymbol{c}}$ occurrences in $\boldsymbol{c}$,
- $\mathrm{R}_{\mathrm{N}}^{\mathrm{P}}$: ROUGE-N precision score.

Finally, ROUGE-N F_1 scores are computed as follow:

$$\mathrm{R}_{\mathrm{N}} = \frac{1}{\frac{\alpha}{\mathrm{R}_{\mathrm{N}}^{\mathrm{P}}} + \frac{1-\alpha}{\mathrm{R}_{\mathrm{N}}^{\mathrm{R}}}} \tag{2.3}$$

where:

- $\alpha = 0.5$,
- R_N^R: ROUGE-N recall score as per equation 2.1,
- R_N^P: ROUGE-N precision score as per equation 2.2,
- R_N: ROUGE-N F_1 score.

2.3.1.2 ROUGE-L: Longest Common Subsequence

ROUGE-L is a measure based on the Longest Common Subsequence (LCS) of words between a candidate summary $\mathbb{C}$ and reference summary(ies) $\mathbb{R}$.

A subsequence is a sequence that can be derived from another sequence by deleting some or no elements without changing the order of the remaining elements. For example, sequence w_1, w_3, w_5 is a subsequence of w_1, w_2, w_3, w_4, w_5 obtained after removing w_2 and w_4. An advantage of using subsequence matches is that they reflect sentence-level word order better than strict consecutive matches.

Computing ROUGE-L requires calculating union LCS scores. For example, given sentence $\boldsymbol{a} = w_1, w_2, w_3, w_4, w_5$ and a set of sentences $\mathbb{B}$ containing $\boldsymbol{b}^{(1)} = w_1, w_2, w_6, w_7, w_8$ and $\boldsymbol{b}^{(2)} = w_1, w_3, w_8, w_9, w_5$, then the LCS of $\boldsymbol{a}$ and $\boldsymbol{b}^{(1)}$ is w_1, w_2 and the LCS of $\boldsymbol{a}$ and $\boldsymbol{b}^{(2)}$ is w_1, w_3, w_5. In this case, $\boldsymbol{u} = w_1, w_2, w_3, w_5$ is the union LCS between $\boldsymbol{a}$ and $\mathbb{B}$ and the union LCS score of $\boldsymbol{a}$ and $\mathbb{B}$ is $|\boldsymbol{u}|/|\boldsymbol{a}| = 4/5$.

ROUGE-L scores of candidate summary $\mathbb{C}$ with respect to reference summary(ies) $\mathbb{R}$ are computed as follows:

$$R_L^R = \frac{\sum_{\boldsymbol{r} \in \mathbb{R}} LCS_\cup(\boldsymbol{r}, \mathbb{C})}{m} \tag{2.4}$$

$$R_L^P = \frac{\sum_{\boldsymbol{r} \in \mathbb{R}} LCS_\cup(\boldsymbol{r}, \mathbb{C})}{n} \tag{2.5}$$

$$\mathrm{R_L} = \frac{1}{\frac{\alpha}{\mathrm{R_L^P}} + \frac{1-\alpha}{\mathrm{R_L^R}}} \tag{2.6}$$

where:

- $\mathbb{C}$: sentences from candidate summary,
- n: total number of words in $\mathbb{C}$,
- $\mathbb{R}$: sentences from the reference summary(ies) for $\mathbb{C}$,
- $\boldsymbol{r}$: sentence from $\mathbb{R}$,
- m: total number of words in $\mathbb{R}$,
- $LCS_{\cup}(\boldsymbol{r}, \mathbb{C})$: union LCS score of $\boldsymbol{r}$ and $\mathbb{C}$,
- $\mathrm{R_L^R}$: ROUGE-L recall score,
- $\mathrm{R_L^P}$: ROUGE-L precision score,
- $\mathrm{R_L}$: ROUGE-L F_1 score,
- $\alpha = 0.5$.

2.3.1.3 Limitations

ROUGE limitations have been pointed out many times over the years, the principal one being that the strict lexical matches on which it is based inherently fails to capture synonymous concepts (Ganesan, 2018; Ng & Abrecht, 2015; ShafieiBavani, Ebrahimi, Wong & Chen, 2018; Zhou, Lin, Munteanu & Hovy, 2006). This bias in favour of surface lexical similarity is especially unfair in the case of summaries making heavy usage of paraphrasing, such as abstractive summaries (Ng & Abrecht, 2015).

Although many extensions to ROUGE have been proposed in order to overcome its limitations (Ganesan, 2018; Ng & Abrecht, 2015; ShafieiBavani *et al.*, 2018; Zhou *et al.*, 2006), none of

the new metrics have gained sufficient traction to dethrone the original ROUGE metrics as the de-facto standard in automatic evaluation of summaries (Kryściński *et al.*, 2019; Owczarzak *et al.*, 2012).

2.3.2 Output length control evaluation

Output length control capability is evaluated using Var_t (Liu *et al.*, 2018) and $\%over$ (Makino, Iwakura, Takamura & Okumura, 2019). Var_t is the variance of summary lengths l_k w.r.t. target length l_t:

$$Var_t = 0.001 \times \frac{1}{p} \sum_{k=1}^{p} |l_k - l_t|^2 \tag{2.7}$$

where p is the number of summaries in the set of candidates $\hat{\mathbb{Y}}$. The function of the 0.001 factor is simply to scale down the resulting metric to more readable values. A low Var_t indicates a high level of control over the length of the output. The $\%over$ metric represents the percentage of summaries longer than the target length in a given set of candidate summaries. A low $\%over$ indicates that a model rarely generates summaries longer than the target length.

2.4 Summary

Training abstractive summarization models in a supervised fashion has started to be possible with the recent apparition of large scale summarization datasets such as CNN/DailyMail (Hermann *et al.*, 2015) and XSum (Narayan *et al.*, 2018). Few aligned datasets such as these are freely available to the public because they are expensive to produce. On the other hand, corpora such as CC-News (Nagel, 2016) provide cheap access to large amounts of news, but can only be used for unsupervised training because they do not provide summaries for the articles they contain.

Unsupervised RNN-based models from chapter 3 are trained on a version CNN/DailyMail dataset that treats summaries as input documents, effectively doubling the size of the corpus.

Unsupervised Transformer-based models from chapter 4 are trained on the CC-News dataset. Two different versions of the corpus are used: CC-News$_{[15;512]}$, which only contains documents

between 15 and 512 tokens, and CC-News$_{[50;512]}$, which only contains documents between 50 and 512 tokens. Documents under 50 tokens were excluded from CC-News$_{[50;512]}$ in an attempt to reduce noise in the corpus.

ROUGE (Lin, 2004; Lin & Hovy, 2003) is the standard metric in automatic evaluation of summaries (Kryściński *et al.*, 2019; Owczarzak *et al.*, 2012), but has well known limitations. Its principal drawback is that it is unable to capture synonymous concepts, which is especially unfair in the case of abstractive summaries (Ganesan, 2018; Ng & Abrecht, 2015; ShafieiBavani *et al.*, 2018; Zhou *et al.*, 2006).

Finally, output length control capability is evaluated using to different metrics: Var_t (Liu *et al.*, 2018), which is the variance of summary lengths l_k w.r.t. target length l_t, and $\%over$ (Makino *et al.*, 2019), which is the percentage of summaries longer than the target length.

CHAPTER 3

UNSUPERVISED ABSTRACTIVE SUMMARIZATION BASED ON RNN

The main idea explored in this chapter is to extend the usage of the LenInit output length control mechanism (section 1.2.7) to train an abstractive summarization model in an unsupervised fashion. The hypothesis is that using this output control method, the encoder can learn to map input sequences to a given vector space while the decoder can learn to map from that same vector space to a sequence of shorter length in a controlled fashion. In hope that the model develops this capability, it is trained to autoencode (section 1.2.3) documents of various lengths (from normal "article lengths" documents to very short, "summary lengths" documents) while its LenInit decoder's target length is set to the correct lengths (the lengths of the documents to autoencode). Once the training is completed, the model is then prompted to generate an output shorter than its input, effectively summarizing it.

The work described in this chapter was conducted in 2018 and 2019. It is during this time period that the Transformer architecture (Vaswani *et al.*, 2017) came to prominence (section 1.3). Unfortunately, the author was not aware of these developments as they were occurring, which is why the model proposed in this chapter is not up to date with the current SOTA. The model proposed in chapter 4 on the other hand is based on the current generation of abstractive summarizers.

3.1 Proposed Model

The proposed model is a basic Encoder-Decoder (section 1.2.4) where the decoder is modified to use the LenInit output length control technique (section 1.2.7). No attention mechanism was used in order to allow for future experiments in multi-document summarization in a similar vein to MeanSum (1.3.2.1). The encoder and decoder components are mLSTMs (Krause *et al.*, 2016) with the following characteristics:

- embedding size: 256,
- number of hidden mLSTM layers: 1,
- hidden layer size: 512.

The vocabulary size is set to 32,000 and the model is pretrained on the Language Modeling (LM) task. The model is trained as an autoencoder using teacher-forcing (Williams & Zipser, 1989) and a dropout rate of 0.1. A standard Cross-Entropy loss (equation 1.6) is used during both pretraining and training phases.

At time of inference, the model is prompted to generate tokens until it outputs a special end-of-sequence tag (</DOC>). Generation is stopped automatically if more than 1000 tokens have been generated for a single summary.

3.2 Experiments

First, a Language Model (LM) was trained on the unfiltered CNN/DailyMail corpus training set (section 2.2.1). Then, a model initialized using the LM's weights was trained on the autoencoding task using the filtered version of the training set. In an attempt to improve performance, training was set to last for 90,000 steps and no early stopping was used. This decision was taken because the loss (error) of the model tended to stabilize quite some time before the prespecified number of training steps was reached. Since there are more than 90,000 items in the training set, then the model was trained for less than one full epoch and overfitting was therefore judged effectively impossible in these conditions.

Three experiments were conducted on this model. In the first experiment, the model was prompted to generate three sets of summaries from the filtered CNN/DailyMail test set's *source* documents: a first set using a target length of 50 tokens, a second set targeting 100 tokens in length and a third set targeting 200 tokens. These three sets of generated summaries were used to evaluate the length control capabilities of the model.

In the second experiment, ROUGE scores (section 2.3.1) were computed for the summaries generated using a target length of 50 tokens. Summaries from the CNN/DailyMail test set were used as reference (section 2.2.1). The goal of this experiment was to evaluate the capacity of the model to effectively summarize news articles.

The goal of the third and last experiment was to evaluate the effect of using a length control mechanism on the quality of the generated texts. For this experiment, a simple RNN encoder-decoder model (without length control mechanism) was trained as a baseline. The baseline model shares the same set of hyperparameters as the experimental model. Furthermore, it's weights were initialized using the same LM's weights and it was trained on the same autoencoding task using the same training set and for the same number of steps. Experimental and baseline models were used to autoencode *summaries* from the filtered CNN/DailyMail test set. ROUGE scores (section 2.3.1) were computed for each generated text using the input (the summaries that were autoencoded) as reference.

3.3 Results

Results for the first experiment can be found in tables 3.1, 3.2 and 3.3 in the form of statistics about summaries generated when targeting lengths of 50, 100 and 200 tokens. Figures 3.1, 3.2 and 3.3 offer visualizations of the same distributions. Table 3.4 shows examples of generated summaries along with the corresponding test set reference summary.

ROUGE scores computed as a result of the second experiment can be found in table 3.5. These scores represent the capacity of the model to effectively summarize news articles.

Finally, ROUGE F_1 scores computed as a result of the third experiment can be found in table 3.6.

3.4 Discussion

Tables 3.1, 3.2 and 3.3 show that the mean length of the generated summaries move according to the target, but are generally longer than desired. This is at least somewhat encouraging as it

Table 3.1 Distributions of summary lengths (target: 50 tokens)

Mean	72
Standard deviation	15
Minimal length	16
First quartile	63
Median	69
Third quartile	76
Maximal length	252

Table 3.2 Distributions of summary lengths (target: 100 tokens)

Mean	121
Standard deviation	38
Minimal length	25
First quartile	102
Median	112
Third quartile	129
Maximal length	1000

suggests that the model is able to learn to control the length of its output to some degree. The fact that the mean length of the generated summaries is longer than the target in all three cases could be explained by the length distributions of the generated texts. Indeed, figures 3.1, 3.2 and 3.3 show that the distributions tend to be highly skewed towards longer documents.

Table 3.3 Distributions of summary lengths (target: 200 tokens)

Mean	219
Standard deviation	90
Minimal length	3
First quartile	160
Median	193
Third quartile	251
Maximal length	1000

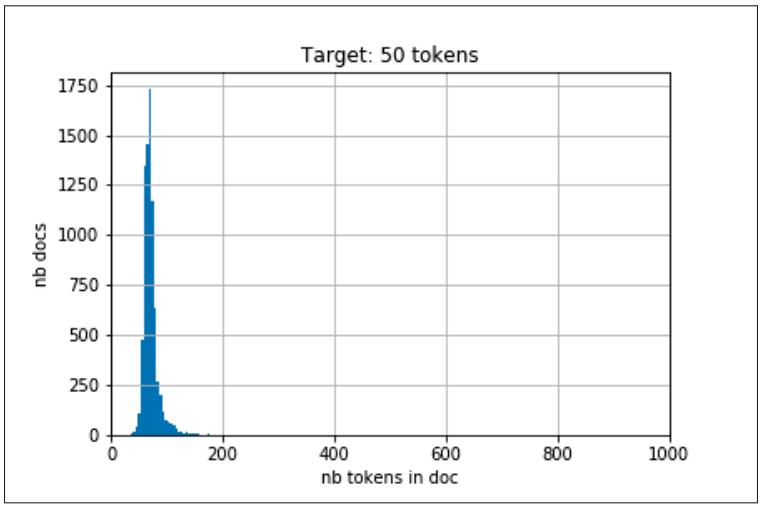

Figure 3.1 Distributions of summary lengths (target: 50 tokens)

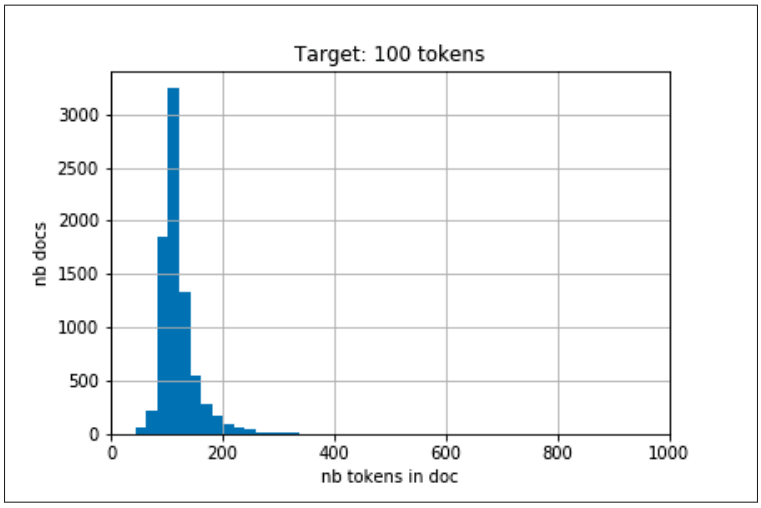

Figure 3.2 Distributions of summary lengths (target: 100 tokens)

Another takeaway from the same tables and figures is that the model seems to have more difficulties controlling the length of its output as the target gets larger. One possible cause for this behavior could be that the corpus used has a high ratio of short documents (half the documents in the training corpus are of summary length, as described in section 2.2.1). Training

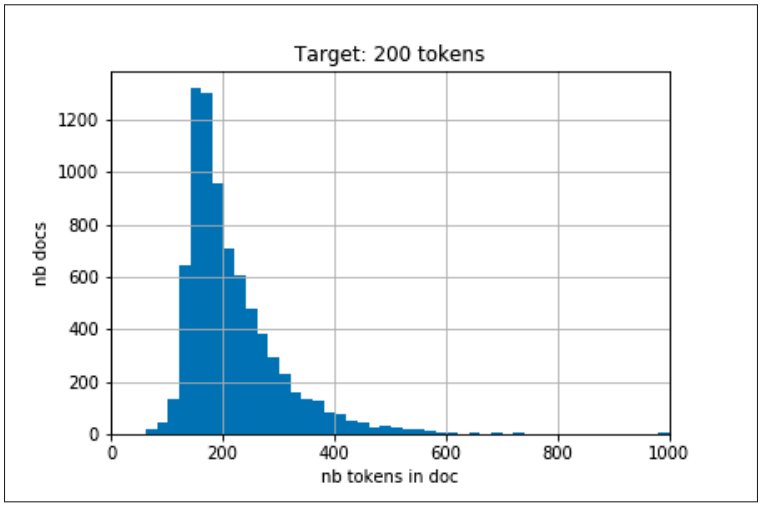

Figure 3.3 Distributions of summary lengths (target: 200 tokens)

on a corpus which is highly skewed toward a particular length of documents could be preventing the model from effectively learning to control its output's length.

Although the generated summaries are readable, they suffer from many of the issues associated with summaries generated using simple encoder-decoder architectures, such as incorrect informational content and unnecessary repetitions (table 3.4). The low ROUGE scores found in table 3.5 also suggest poor summarizing capabilities.

The ROUGE F_1 scores from table 3.6 show that the LenInit output control mecanism has a significant negative effect on the quality of the generated texts, causing a drop of 5.1 ROUGE-1 points (17%), 1.72 ROUGE-2 points (32%) and 4.61 ROUGE-L points (17%). One possible cause for this effect is the repurposing of the decoder's memory cell by the control mecanism, effectively adding responsibilities to the cell without increasing its capacity.

3.5 Future work

Future experiments planned in hope of improving output length control in the context of unsupervised RNN-based abstractive summarization include training on a version of the CC-

Table 3.4 Examples of generated summaries along with the corresponding test set reference summary

Reference	Membership gives the ICC jurisdiction over alleged crimes committed in Palestinian territories since last June. Israel and the United States opposed the move, which could open the door to war crimes investigations against Israelis.
Target length	**Generated summary**
50	(CNN)A South Korean prison camp in the country's largest city was hijacked by a group of army soldiers in the past year. The United Nations has been trying to resolve the crisis with the country's death penalty in recent years. As the first of the last two years, it was not immediately clear how much the North is, and it is not just a case of a war.
100	(CNN)A South Korean prison camp in the country's largest city was hijacked by a group of army soldiers in the past year, killing all four in the last two years, a spokesman said. The group, which is to be in the country, is the only country to be in the country. The country's president is the first of its kind to be buried in the United States. A number of U.N. officials have ruled out a "very serious" case for the country's war, and could not be used in the region.
200	(CNN)A South Korean prison camp in the country's capital was left in the hands of the army in the past, but it was not immediately known how many of the other prisoners were killed. The group, which is to be the first country to be the only country to be released, is one of the most endangered North Africa. The United States has been the target of a series of bombings that killed three people, including two of the two soldiers in the country's North African country. The United States, which has been fighting for the death of a single-war hero, has been sentenced to death for the death of a former leader. The court said it was not a "very serious" decision, and the verdict will be added to the case. The United States will not be able to hold the death penalty against the island's president, who will be sentenced next month.

Table 3.5 ROUGE scores of summaries generated using a target length of 50 tokens. Summaries from the CNN/DailyMail test set were used as reference

Metric	**Recall**	**Precision**	**F_1**
ROUGE-1	23.19	18.42	19.93
ROUGE-2	2.73	2.16	2.34
ROUGE-L	21.01	16.64	18.04

News corpus (Nagel, 2016) filtered to produce a dataset free of strong bias toward any particular document length.

Table 3.6 ROUGE F_1 scores of the RNN baseline and LenInit models computed on autoencoded texts from the test set

Metric	Baseline RNN	LenInit RNN
ROUGE-1	29.44	24.34
ROUGE-2	5.39	3.67
ROUGE-L	26.82	22.21

In order to diminish the negative effect of the output control mechanism on the quality of the generated texts, other techniques could be tested, such as adding a dedicated output control cell to the decoder LSTM (instead of repurposing the memory cell as with LenInit) or using a completely different mechanism, such as LenEmb (section 1.2.7).

Future experiments planned in hope of improving text quality include using a bidirectionnal LSTM (Schuster & Paliwal, 2010) instead of a mLSTM (Krause *et al.*, 2016) in the encoder and adding an attention mechanism (section 1.2.5) to help the model process long input documents. Also, adding a pointer-generator network (See *et al.*, 2016) could help the model handle out-of-vocabulary terms and generate summaries more convincingly connected to their input documents.

Finally, experiments in multi-document summarization using a training regime similar to MeanSum (section 1.3.2.1) are also planned. However, given that MeanSum decodes a summary from a code vector obtain by computing the mean of multiple documents' code vectors, these experiments would first exclude the usage of an attention mechanism.

3.6 Summary

In summary, the observation that the mean and median length of the generated summaries are close to and move according to the target suggest that the proposed unsupervised RNN-based model is indeed able to control its output length to some degree, however, the level of control seems to diminish as the target length increases. This behavior might be caused by training on a corpus skewed toward short documents. More importantly, the informational content of the

summaries generated by the unsupervised RNN-based model is often inaccurate with respect to the input document. This inaccuracy can be in part attributed to the simplicity of model, but it has been observed that the LenInit output control mechanism itself also negatively affect the quality of the generated texts.

CHAPTER 4

UNSUPERVISED ABSTRACTIVE SUMMARIZATION BASED ON TRANSFORMER

The ideas explored in this chapter include to extend the usage of output length control mechanisms (section 1.2.7) to produce an abstractive summarization model by training it on the autoencoding task. The intuition behind this proposal is that explicitly providing the correct length of the output during training discourages an autoencoder from learning to encode this information. Instead, the encoder is encouraged to learn to map input sequences to a latent vector representation space disentangled from output length considerations while the decoder learns to rely on the explicitly provided target value to generate sequences of correct lengths. In addition to experiments on the autoenconding task, experiments on the *denoising* autoencoder task (Vincent *et al.*, 2008) are also conducted.

This second take on experiments from chapter 3 was motivated by the poor results of the first attempt and the rise a new generation of SOTA systems based on the Transformer architecture (Vaswani *et al.*, 2017) that occured in the mean time (section 1.3). The difference with experiments described in chapter 3 is that instead of using an RNN-based architecture (section 1.2.2) and the LenInit control mechanisim (Kikuchi *et al.*, 2016), it is a Transformer-based architecture and the LDPE (or LRPE) control mechanism (Takase & Okazaki, 2019) that are used.

Another idea explored in this chapter is to take advantage of the lead bias in news articles (Kedzie *et al.*, 2018; See *et al.*, 2016; Zhang *et al.*, 2019) to generate an aligned summarization dataset from a large, unaligned news corpus such as CC-News (Nagel, 2016). The lead biais is the result of journalistic conventions which encourage news writers to divulge the most important information near the beginning of an article. The strategies to generate an aligned dataset investigated in this chapter revolve around the idea of using the firsts n sentences of a news article as its target summary.

4.1 Proposed Model

The proposed Unsupervised Abstractive Summarization model, or UASum, consists of a BertSum encoder (Liu & Lapata, 2019) coupled with a length control enabled Transformer decoder. Experiments were conducted using the LDPE, LRPE, LDPE+PE and LRPE+PE control mechanisms (Takase & Okazaki, 2019). The model is trained to denoise and autoencode each document as follow:

$$\boldsymbol{X}_T^{(i)} = \phi_D(\phi_E(\boldsymbol{X}_S^{(i)}), t_i) \tag{4.1}$$

where $\boldsymbol{X}_S^{(i)}$ is a preprocessed Source sequence, ϕ_E the model's encoder, ϕ_D the model's decoder and $\boldsymbol{X}_T^{(i)} = \boldsymbol{Y}^{(i)}$ the corresponding preprocessed Target sequence. Experiments were conducted using X, Y and Z types of noise as described in section 2.2.2. Target length t_i is set to the length of the target sequence $\boldsymbol{X}_T^{(i)}$ during training.

Once training is completed, the model can be prompted to generate summaries as follow:

$$\hat{\boldsymbol{Y}} = \phi_D(\phi_E(\boldsymbol{X}), t) \tag{4.2}$$

where input document $\boldsymbol{X}$ has no artificial noise added to it and $\hat{\boldsymbol{Y}} = \hat{\boldsymbol{y}}_0, \ldots, \hat{\boldsymbol{y}}_{t'}$ is the corresponding generated summary of length $t' \approx t$.

The BertSum encoder has 12 layers, a hidden size of 768 and 12 self-attention heads. Its feed-forward layers have a hidden size of 3072 and its weigths are initialized using the *bert-base-uncased*[1] version of Bert. The decoder is a randomly initialized standard Transformer decoder with 6 layers, a hidden size of 768 and 8 self-attention heads. Its feed-forward layers have a hidden size of 2048.

Following (Liu & Lapata, 2019), encoder and decoder use separate optimizers and learning rate schedules. The encoder uses an Adam optimizer (Kingma & Ba, 2017) with $\beta_1 = 0.9$ and a learning rate schedule defined by $lr_E = \widetilde{lr}_E \cdot min(step^{-0.5}, step \cdot warmup_E^{-1.5})$ with $\widetilde{lr}_E = 2e^{-3}$

[1] https://git.io/fhbJQ

and $warmup_E = 20,000$. The decoder uses a second Adam optimizer with $\beta_2 = 0.999$ and a learning rate schedule defined by $lr_D = \widetilde{lr}_D \cdot min(step^{-0.5}, step \cdot warmup_D^{-1.5})$ with $\widetilde{lr}_D = 0.1$ and $warmup_D = 10,000$. Dropout with probability 0.1 is applied before all linear layers and label smoothing (Szegedy, Vanhoucke, Ioffe, Shlens & Wojna, 2016) with smoothing factor 0.1 is also used. All models were trained for 200, 000 steps. Backward propagation is performed every 5 steps using gradient accumulation.

Model checkpoints are saved and evaluated on the validation set every 2, 000 steps. Model selection is done by choosing the checkpoint which produces the lowest perplexity score on the validation set. Decoding on the validation and test sets is performed using beam search with a beam size of 5 and an α of 0.95 for length penalty (Wu *et al.*, 2016). During test, decoding continues until an end-of-sequence token is emitted or the candidate summary reached a maximum length of 200 tokens. Repeated trigrams are blocked to reduce redundancy. This is done by setting $p(\hat{\boldsymbol{y}}_k) = 0$ during beam search if outputting $\boldsymbol{y}_k$ would create a trigram already present in the current beam (Paulus, Xiong & Socher, 2018).

The proposed model was implemented using PyTorch (Paszke, Gross, Massa, Lerer, Bradbury, Chanan, Killeen, Lin, Gimelshein, Antiga, Desmaison, Kopf, Yang, DeVito, Raison, Tejani, Chilamkurthy, Steiner, Fang, Bai & Chintala, 2019) and OpenNMT (Klein, Kim, Deng, Senellart & Rush, 2017).

4.2 Experiments

Two sets of experiments were conducted using the Transformer-based architecture.

The first set of experiments focused on analysing the performance cost of adding an output length control mechanism to the BertSumExtAbs model (Liu & Lapata, 2019) in the context of supervised training. Experiments were conducted using 4 variants of output-length-control mechanisms: LDPE, LRPE, LDPE+PE and LRPE+PE (section 1.2.7). Target lengths of 10, 25, 60 and 160 tokens were used. All models were built using the same BertSum encoder which was first pretrained on the extractive summarization task. Pretraining and fine-tuning

on the abstractive summarization task were performed using the CNN/DailyMail dataset. The preciseness of the different output length control mechanisms is assessed and the performances of the control-enabled BertSumExtAbs models are compared to the original model.

The second set of experiments focused on studying the effect of different preprocessing strategies in the context of unsupervised training of abstractive summarizers. Among the explored strategies, *Text Infilling*, *Sentence Permutation* and *Extract-N* (section 2.2.2) involve injecting different forms of noise in the input documents, while training the model to regenerate a prespecified, noise free text. The *Remove-N* preprocessing strategy on the other hand generates an aligned summarization dataset from a large, unaligned news corpus ($\mathbb{X} \rightarrow \mathbb{X}_S, \mathbb{X}_T = \mathbb{Y}$) by splitting each document so that the firsts *n* sentences of a news article are used as target summary and the rest of the article is used as input document (section 2.2.2).

A subset of combinations of strategies was also tested: *Extract-N + Text Infilling + Sentence Permutation* and *Remove-N + Text Infilling + Sentence Permutation*.

4.2.1 Unsupervised Experiments' Baseline Models

To the best of the authors' knowledge, UASum was the first attempt at unsupervised single-document abstractive summarization of news at the time the experiments reported here were conducted, and there were therefore no available benchmark on the CNN/DailyMail or XSum datasets. The baselines described below were used instead.

Oracle: An Oracle system (Liu & Lapata, 2019; Nallapati, Zhai & Zhou, 2017) is used as an upperbound reference. Oracle summaries are generated by selecting sentences which maximize the ROUGE-2 score against the reference summary.

Supervised baselines: Bart (Lewis *et al.*, 2019) is used as a reference for state-of-the-art abstractive single-document summarization. It is a Tranformer-based encoder-decoder that combines a Bert-style bidirectional encoder (Devlin *et al.*, 2018) and a Gpt-style, left-to-right

decoder (Radford, Narasimhan, Salimans & Sutskever, 2018; Radford *et al.*, 2019). It is pretrained as a denoising autoencoder and then fine-tuned on the abstractive summarization task.

Unpaired baselines: Wang & Lee (2018) borrow ideas from adversarial autoencoders (Makhzani, Shlens, Jaitly, Goodfellow & Frey, 2016) and Cycle GAN (Zhu, Park, Isola & Efros, 2017) to propose a model that learns to summarize documents using large datasets of *unpaired* documents and summaries. The model is composed of an autoencoder and a discriminator network that uses sentences from the set of reference summaries during training.

Unsupervised baselines (extractive): A LEAD-3 baseline (Nallapati *et al.*, 2017) is used as a minimal-effort reference for single-document summarization. LEAD-3 generate summaries by selecting the first 3 sentences of a document.

PACSUM BERT (Zheng & Lapata, 2019) is used as a reference for state-of-the-art unsupervised extractive summarization. It uses BERT to capture sentential meaning and compute sentence similarity for the purpose of sentence selection.

4.3 Results and Discussion

Output length control results are shown in table 4.1. From these results it becomes apparent that LDPE provides the most consistently precise output length control among the tested variants. Results using a target length of 10 tokens in particular demonstrate that it is the only option that does not dramatically overshoot in that range.

Table 4.2 shows ROUGE F_1 scores for the 4 same variants of BERTSUMEXTABS summarizers. Target length was set to 60 tokens. Consistent with output-length-control evaluation results, the LDPE variant is shown to produce the highest scores. Although automatic evaluation rates BERTSUMEXTABS LDPE slightly lower than an unmodified BERTSUMEXTABS reference, an ad-hoc evaluation of the generated summaries did not reveal any significant drop in text quality. One possible explanation for the difference is that a tight control of the output's length could produce lower ROUGE F_1 scores when the reference summary is significantly shorter or longer

Table 4.1 Var_t and $\%over$ results for 4 variants of output-length-control enabled BertSumExtAbs summarizers. Average length Avg_t is also provided

Model	***Avg*$_t$**	***Var*$_t$**	***%over***
Target: 10 tokens			
BertSumExtAbs LDPE	**9**	**0.003**	**12.98**
BertSumExtAbs LDPE+PE	134	18.74	94.14
BertSumExtAbs LRPE	13	0.011	90.16
BertSumExtAbs LRPE+PE	19	0.104	99.82
Target: 25 tokens			
BertSumExtAbs LDPE	23	**0.005**	**2.16**
BertSumExtAbs LDPE+PE	**25**	0.138	16.95
BertSumExtAbs LRPE	23	0.015	25.49
BertSumExtAbs LRPE+PE	28	0.037	67.386
Target: 60 tokens			
BertSumExtAbs LDPE	**58**	**0.005**	**1.78**
BertSumExtAbs LDPE+PE	64	0.400	32.07
BertSumExtAbs LRPE	**58**	0.007	14.62
BertSumExtAbs LRPE+PE	48	0.187	4.09
Target: 160 tokens			
BertSumExtAbs LDPE	154	**0.261**	1.38
BertSumExtAbs LDPE+PE	**163**	1.592	63.30
BertSumExtAbs LRPE	145	0.297	**0.05**
BertSumExtAbs LRPE+PE	98	3.919	0.044

than the target length. Table 4.3 shows example summaries generated by the BertSumExtAbs LDPE model.

Table 4.2 ROUGE F_1 scores on the CNN/DailyMail test set for 4 variants of output-length-control enabled BertSumExtAbs summarizers. Target length was set to 60 tokens

Model	**R1**	**R2**	**RL**
Bart	**44.16**	**21.28**	**40.90**
BertSumExtAbs	41.88	19.42	38.93
BertSumExtAbs LDPE	**40.27**	**18.46**	**37.59**
BertSumExtAbs LDPE+PE	38.55	17.18	35.84
BertSumExtAbs LRPE	40.13	18.44	37.43
BertSumExtAbs LRPE+PE	37.75	17.08	35.10

Table 4.3 Examples of generated summaries generated by BERTSUMEXTABS LDPE

Reference	cctv camera captured a man in a hoodie dragging woman 's limp body from car parked at 131st street and jamaica avenue in queens saturday . woman was left slumped on the sidewalk for 20 minutes until paramedics arrived and took her to a hospital . she was placed on a ventilator and remains in critical but stable condition with leg injuries
BERTSUMEXTABS	the nypd released the video tuesday , along with a photo of the victim in the hospital in hopes of identifying her . the woman was critically injured and still has not regained consciousness . the video , obtained from a store owner in the richmond hill section of queens , shows a man dressed in a hooded sweatshirt and sweatpants dragging the limp body of a woman from the backseat of a parked car
Target length	**BERTSUMEXTABS LDPE generated summary**
60	new york city police are searching for a man who was caught on a surveillance camera dumping an unconscious woman on a street in queens . the woman was critically injured and still has not regained consciousness . the video shows a man dragging the body from the backseat of a parked car at around 12.30 am
25	new york city police are searching for a man who was caught on a surveillance camera dumping an unconscious woman on a street in queens
10	new york city police are searching for a man

Output-length-control experiments were first conducted on 5 variants of Unsupervised Abstractive Summarizers (UASUM). Target lengths of 10, 25, 60 and 160 tokens were used. All variants were built using the same BERTSUM encoder. No pretraining on the extractive summarization task was performed. All models were trained on the basic autoencoding task using the CC-News$_{[50;512]}$ dataset, with the exception of UASUM$_{[15;512]}$ LDPE which was trained on the CC-News$_{[15;512]}$ dataset. Results are shown in table 4.4. As with supervised experiments, LDPE is shown to provide the most consistently precise output length control among the tested variants. One limitation that becomes apparent is the difficulty that models have to generate texts of lengths they did not encountered during training. Results using a target length of 10 tokens in particular demonstrate that no variant was able to consistently generate sequences in that range.

Table 4.4 Var_t and $\%over$ results for 5 variants of UASum models. Average length Avg_t is also provided

Model	Avg_t	Var_t	$\%over$
Target: 10 tokens			
UASum$_{[15;512]}$ LDPE	**50**	3.261	**93.22**
UASum LDPE	56	**2.393**	100.00
UASum LDPE+PE	120	14.96	99.32
UASum LRPE	150	21.84	99.99
UASum LRPE+PE	56	2.497	100.00
Target: 25 tokens			
UASum$_{[15;512]}$ LDPE	**26**	0.051	34.50
UASum LDPE	**24**	**0.003**	**4.82**
UASum LDPE+PE	61	3.807	83.86
UASum LRPE	146	16.75	99.99
UASum LRPE+PE	54	0.950	99.96
Target: 60 tokens			
UASum$_{[15;512]}$ LDPE	**59**	0.005	1.67
UASum LDPE	**59**	**0.004**	**1.38**
UASum LDPE+PE	69	0.387	71.82
UASum LRPE	163	11.90	99.72
UASum LRPE+PE	97	1.672	98.67
Target: 160 tokens			
UASum$_{[15;512]}$ LDPE	154	0.303	**4.45**
UASum LDPE	**155**	**0.197**	6.40
UASum LDPE+PE	129	1.634	5.80
UASum LRPE	187	1.502	86.77
UASum LRPE+PE	172	1.057	71.95

Disappointingly, an analysis of the summaries generated by the 5 variants of UASum trained on the basic autoencoding task revealed that they mostly learned to regenerate input documents up to the target length. This phenomenon can be observed in the example summaries from table 4.5.

Further experiments were designed in order to prevent the unsupervised models from simply learning the identity function. Experiments involving training on the denoising autoencoder task unfortunately did not fare better. However, training on a dataset preprocessed using the *Remove-N* strategy did provide some measure of success. Table 4.6 shows an example representative of

Table 4.5 Examples of summaries generated by UASUM[15;512] LDPE

Input document	(cnn) when bruce jenner told abc 's diane sawyer and the world on friday night that " yes , for all intents and purposes , i 'm a woman , " the declaration was n't par-ticularly surprising . . paparazzi have been stalking the former olympic champion for months , breathlessly documenting his increasingly feminine appearance down to his lip gloss , pierced ears and french manicure . a cheer , a shrug – and not much else . except the occasional prod at jenner also being republican . of course , jenner 's family owned the moment , being the reality tv mavens they are – the jenners and the kardashians . " so very proud of you , my hero , " tweeted daughter kendall jenner . his other daughter kylie jenner expressed mixed emotions . " understandingly , this has been very hard for me . you will hear what i have to say when i 'm ready to but ... this is n't about me . (cont...)
Reference	social media largely supports jenner . more people seemed intrigued that he 's a republican

Target length	**UASUM[15;512] LDPE generated summary**
60	(cnn) when bruce jenner told abc 's diane sawyer and the world on friday night that " yes , for all intents and purposes , i 'm a woman , " the declaration was n't par-ticularly surprising . . paparazzi have been stalking the former olympic champion for months
25	(cnn) when bruce jenner told abc 's diane sawyer and the world on friday night that " yes , for all

summaries generated by a UASUM LDPE model trained on a version of the CC-News[50;512] dataset preprocessed according to the *Remove-N* strategy[2].

It can be observed from this example that the model does not simply learn the identity function and instead demonstrate the capacity to form new sentences summarizing information contained at different locations of the input document. However, it is also apparent that the model inserts a fair amount of noise at the start of its summaries. An ad-hoc inspection revealed that about 50% of the generated summaries consist of noise, usually taking the form of a request for subscription or other types of advertisement. This is most likely the result of having a large portion of documents from the CC-News training datasets containing similar advertisements due to an insufficient cleanup strategy.

[2] *n* was set to 3 for all experiments.

Table 4.6 Summary generated by a UASum LDPE model trained on a version of the CC-News$_{[50;512]}$ dataset preprocessed according to the *Remove-N* strategy

Input document	nestled between a 65 ft high snow cavern , the winding track of murodo looks like track that could be on mario kart . however this steep - walled phenomena is located in japan and attracts visits from thousands of tourists annually - with the number predicted to rise this year . the tateyama kurobe alpine route opened to the public on today and allows tourists to view the spectacular snow - walled passageway along the 1000 ft section . the tateyama kurobe alpine route opened for visitors today , with tourists marvelling at the 65 ft high walls the murodo - daira area of tateyama unsurprisingly experiences one of the heaviest snowfalls in the world , with the 65 ft walls being created every year , and then melting by july . the impressive route opened in 1971 and usually draws about a million visitors every year although numbers have failed to reach that figure in the last few years . an estimated 910 , 00 0 flock to the frozen passage last year . (cont.)
Reference	tateyama kurobe alpine route opened to the public on today allowing guests to marvel at the colossal snow walls. the 1000ft section can be walked by visitors , and usually draws a million tourists every year. the snowy section is part of a 37km route , with sights such as kurobe dam and the hida mountains on the way
Target length	**UASum LDPE generated summary**
60	get daily updates directly to your inbox subscribe thank you for subscribing !. could not subscribe , try again later invalid email. the tateyama kurobe alpine route opened for visitors today , with tourists marvelling at the 65ft high walls being created every year .

Table 4.7 shows ROUGE F_1 scores for UASum LDPE models trained on two variants of the CC-News$_{[50;512]}$ dataset. The *Remove-N* variant was trained on a CC-News$_{[50;512]}$ dataset preprocessed according to the *Remove-N* strategy by itself. The *RN+TI+SP* variant was trained on a CC-News$_{[50;512]}$ dataset preprocessed using a combination of the *Remove-N*, *Text Infilling* and *Sentence Permutation* strategies. As with all other Transformer-based experiments in this work, testing was conducted using the CNN/DailyMail test subset. Target length was set to 60 tokens.

As table 4.7 shows, the model trained on a dataset preprocessed using a combination of the *Remove-N*, *Text Infilling* and *Sentence Permutation* strategies performs slightly better than the model trained on a dataset preprocessed using the *Remove-N* strategy by itself.

Table 4.7 ROUGE F_1 scores on the CNN/DailyMail test set for 2 variants of UASum models. Target length was set to 60 tokens. Oracle and Lead-3 results were obtained from (Liu & Lapata, 2019). PacSum Bert results were obtained from (Zheng & Lapata, 2019). WGAN and REINFORCE GAN results were obtained from (Wang & Lee, 2018)

Model	**R1**	**R2**	**RL**
Oracle	52.59	31.24	48.87
Unsupervised baselines (extractive)			
Lead-3	40.42	17.62	36.67
PacSum Bert	**40.7**	**17.8**	**36.9**
Unpaired baselines			
WGAN	35.14	**9.43**	**21.04**
REINFORCE GAN	**35.51**	9.38	20.98
UASum LDPE			
Remove-N	20.70	6.23	18.89
RN+TI+SP	21.29	6.4	19.41

4.4 Future work

The example from table 4.6 exposes the need for a more robust corpus cleanup strategy than the simple filtering detailed in section 2.2.2. Given the noise ratio observed in the generated summaries, it very likely that this would result in a significant boost of ROUGE scores as reported in table 4.7.

This hypothesis was actually confirmed during the redaction of the report when a model named Ted (Yang *et al.*, 2020) was published in preprint. Ted is another model based on the Transformer architecture that essentially uses the *Remove-N* strategy on a carefully cleaned dataset. Table 4.8 displays the ROUGE F_1 scores obtained in this manner.

Table 4.8 TED 10L8H (Yang *et al.*, 2020) ROUGE F_1 scores on the CNN/DailyMail test set

Model	R1	R2	RL
TED 10L8H	38.73	16.84	35.40

It should be noted that TED does not possess output length control capabilities.

4.5 Summary

In summary, experiments in this Chapter confirmed the usability of the output length control mechanisms proposed by Takase & Okazaki (2019) in the context of news summarization, with the LDPE strategy offering the highest precision of control under both supervised and unsupervised training.

Multiple unsupervised training strategies were tested, most of which resulted in models disappointingly learning the identity function. One notable exception is the *Remove-N* strategy. However, an ad-hoc analysis of summaries generated by models trained on datasets preprocessed using this strategy revealed that they contain a high ratio of noise, which in turn help identify the shortcoming of the CC-News corpus cleanup procedure.

The soundness of the *Remove-N* strategy was confirmed in parallel to the work described in this report when a model named TED (Yang *et al.*, 2020) was published in preprint.

CONCLUSION

This work constitute the first step in a large R&D project conducted by Canadian company Croesus. The aim of the project is to automate the generation of reports explaining the performance of financial portfolios over a given period of time. The goal of this first step was to start evaluating the feasibility of the project.

The first contribution of this work is the development of a data-to-text architecture designed to model the automatic reporting system (figure 0.1). This high level view represents the system as a pipeline where each stage fulfils a particular role. This has allowed to determine that some stages of the pipeline constitute high risk research problem whereas other stages are more akin to regular software development.

The first research problem to receive attention was to automate the analysis of time series describing a selection of positions from a portfolio in order to identify "significant events". However, work on this subject was quickly abandoned as it became obvious that domain experts were needed so the concept of "significant events" could be correctly defined in the context of automatic portfolio performance reporting. Unfortunately, no such expert was sufficiently available at the time. It was then decided to focus on the problem of unsupervised training of automatic abstractive summarization models with the capacity to control their output's length. Two different architectures of Neural Networks were studied as potential solutions: RNN encoder-decoders and Transformer.

Chapter 3 described the experiments and results using the RNN approach. This type of architecture was eventually discarded due to the poor performance that were achieved and the effort necessary to improve results.

Chapter 4 described the experiments and results using the Transformer approach. These experiments provided the second and third contributions of this work. The second contribution

is that abstractive summarization of controllable length was achieved to a satisfactory level of success in the context of supervised training. To the best of the author's knowledge, the supervised Transformer model proposed in this work is the current SOTA for the task of length-controlled, abstractive summarization of news. The third and last contribution of this work is the progress achieved in the field of unsupervised, output-length-control enabled abstractive summarization. Although it is ultimately the publication in preprint of a model named Ted (Yang *et al.*, 2020) that confirmed the soundness of the proposed unsupervised training strategy, it should be noted that Ted cannot control the length of its output. Future work for this research problem consist of dispelling any remaining doubt by implementing a more robust corpus cleanup strategy and using the resulting dataset to train the proposed unsupervised model.

However, following the success of the output length control experiments conducted in this work, the recent developments in unsupervised abstractive summarization and the general pace at which the field have been evolving in the last two years, Croesus judged that risk with respect to the task of unsupervised abstractive summarization of controllable length was mitigated to a tolerable level. Therefore, work on the data interpretation stage of the pipeline is set to resume soon.

Efforts will now focus on the creation of a "significant events" dataset covering 10 to 20 stocks for a period of time about on year. In order to avoid to have to rely on the direct involvement of domain experts, the data provided by the "Why Is It Moving" (WIIM) API from Benzinga [1] shall be used. This API provides one-sentence explanations about why a stock is trading higher or lower on any given day. Explanations are redacted by a team of analysts based on press releases, news items and SEC filings among other things. The process used to determined what is a significant event is documented in the SLA of the service. The possibility to use the source of data in the creation of an end-to-end evaluation dataset is also being considered.

[1] https://www.benzinga.com/apis/cloud-product/bz-why-is-it-moving/

REFERENCES

Alemu, H. Z., Wu, W. & Zhao, J. (2018). Feedforward Neural Networks with a Hidden Layer Regularization Method. *Symmetry*, 10(10), 525. doi: 10.3390/sym10100525.

Ba, J. L., Kiros, J. R. & Hinton, G. E. (2018, July, 21). Layer Normalization [arXiv e-prints]. Consulted at http://arxiv.org/abs/1607.06450.

Bahdanau, D., Cho, K. & Bengio, Y. (2014, September, 1). Neural Machine Translation by Jointly Learning to Align and Translate [Presented at ICLR 2015]. Consulted at http://arxiv.org/abs/1409.0473.

Baumel, T., Eyal, M. & Elhadad, M. (2018, January, 23). Query Focused Abstractive Summarization: Incorporating Query Relevance, Multi-Document Coverage, and Summary Length Constraints into seq2seq Models [arXiv e-prints]. Consulted at https://arxiv.org/abs/1801.07704v2.

Belz, A. & Reiter, E. (2006, April). Comparing Automatic and Human Evaluation of NLG Systems. *11th Conference of the European Chapter of the Association for Computational Linguistics*. Consulted at https://www.aclweb.org/anthology/E06-1040.

Bengio, Y., Lamblin, P., Popovici, D. & Larochelle, H. (2007). Greedy Layer-Wise Training of Deep Networks. In Schölkopf, B., Platt, J. C. & Hoffman, T. (Eds.), *Advances in Neural Information Processing Systems 19* (pp. 153-160). Vancouver, Canada: MIT Press.

Cauchy, A. (1847). Méthode générale pour la résolution des systèmes d'équations simultanées. *Compte rendu des séances de l'académie des sciences*, 536–538.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. & Bengio, Y. (2014a, October). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1724-1734.

Cho, K., van Merriënboer, B., Bahdanau, D. & Bengio, Y. (2014b, October). On the Properties of Neural Machine Translation: Encoder–Decoder Approaches. *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pp. 103–111. doi: 10.3115/v1/W14-4012.

Chu, E. & Liu, P. J. (2018, October, 12). MeanSum: A Neural Model for Unsupervised Multi-document Abstractive Summarization [arXiv e-prints]. Consulted at http://arxiv.org/abs/1810.05739.

Chu, E. & Liu, P. J. (2019). MeanSum : A Neural Model for Unsupervised Multi-Document Abstractive Summarization. *International Conference on Machine Learning*, pp. 1223–1232. Consulted at http://proceedings.mlr.press/v97/chu19b/chu19b.pdf.

Das, D. & Martins, A. F. (2007). A Survey on Automatic Text Summarization. Consulted at https://www.cs.cmu.edu/~nasmith/LS2/das-martins.07.pdf.

Devlin, J., Chang, M.-W., Lee, K. & Toutanova, K. (2018, October, 10). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [arXiv e-prints]. Consulted at http://arxiv.org/abs/1810.04805.

Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science*, 14(2), 179-211.

Erkan, G. & Radev, D. R. (2004). LexRank: Graph-based Lexical Centrality as Salience in Text Summarization. *Journal of Artificial Intelligence Research*, 22, 457–479. doi: 10.1613/jair.1523.

Fan, A., Grangier, D. & Auli, M. (2018, July). Controllable Abstractive Summarization. *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pp. 45–54. doi: 10.18653/v1/W18-2706.

Fevry, T. & Phang, J. (2018). Unsupervised Sentence Compression using Denoising Auto-Encoders. *Proceedings of the 22nd Conference on Computational Natural Language Learning (CoNLL)*, pp. 413–422. doi: 10.18653/v1/K18-1040.

Ganesan, K. (2018). ROUGE 2.0: Updated and Improved Measures for Evaluation of Summarization Tasks. Consulted at http://arxiv.org/abs/1803.01937.

Gehring, J., Auli, M., Grangier, D., Yarats, D. & Dauphin, Y. N. (2017). Convolutional Sequence to Sequence Learning. *Proceedings of the 34th International Conference on Machine Learning*, 70, 1243–1252.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014). Generative Adversarial Nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems* (vol. 2, pp. 2672–2680). Consulted at http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep Learning*. MIT Press.

Graff, D. & Cieri, C. (2003, January, 28). English Gigaword LDC2003T05 [Web Download. Linguistic Data Consortium]. Consulted at https://catalog.ldc.upenn.edu/LDC2003T05.

Graham, Y. (2015, September). Re-evaluating Automatic Summarization with BLEU and 192 Shades of ROUGE. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 128-137.

Grusky, M., Naaman, M. & Artzi, Y. (2018, June). Newsroom: A Dataset of 1.3 Million Summaries with Diverse Extractive Strategies. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 708-719.

Hamborg, F., Meuschke, N., Breitinger, C. & Gipp, B. (2017). news-please: A Generic News Crawler and Extractor. *Proceedings of the 15th International Symposium of Information Science*, pp. 218–223. Consulted at https://www.gipp.com/wp-content/papercite-data/pdf/hamborg2017.pdf.

He, K., Zhang, X., Ren, S. & Sun, J. (2016, June). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778. doi: 10.1109/CVPR.2016.90.

Hermann, K. M., Kočiský, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M. & Blunsom, P. (2015, December). Teaching Machines to Read and Comprehend. *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, pp. 1693-1701.

Hirao, T., Yoshida, Y., Nishino, M., Yasuda, N. & Nagata, M. (2013). Single-Document Summarization as a Tree Knapsack Problem. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1515–1520. Consulted at https://www.aclweb.org/anthology/D13-1158.

Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735-1780.

Howard, J. & Ruder, S. (2018, July). Universal Language Model Fine-tuning for Text Classification. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, 1, 328–339. doi: 10.18653/v1/P18-1031.

Jang, E., Gu, S. & Poole, B. (2016, November, 03). Categorical Reparameterization with Gumbel-Softmax [arXiv e-prints]. Consulted at http://arxiv.org/abs/1611.01144.

Ji, J., Wang, Q., Toutanova, K., Gong, Y., Truong, S. & Gao, J. (2017, July, 09). A Nested Attention Neural Hybrid Model for Grammatical Error Correction [arXiv e-prints]. Consulted at http://arxiv.org/abs/1707.02026.

Kaiser, L. & Bengio, S. (2016, December). Can Active Memory Replace Attention? *Advances in Neural Information Processing Systems 29*, pp. 3781–3789. Consulted at http://papers.nips.cc/paper/6295-can-active-memory-replace-attention.

Kalchbrenner, N., Espeholt, L., Simonyan, K., van den Oord, A., Graves, A. & Kavukcuoglu, K. (2016, October, 31). Neural Machine Translation in Linear Time [arXiv e-prints]. Consulted at http://arxiv.org/abs/1610.10099.

Kedzie, C., McKeown, K. & Daumé III, H. (2018). Content Selection in Deep Learning Models of Summarization. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Kikuchi, Y., Neubig, G., Sasano, R., Takamura, H. & Okumura, M. (2016, November). Controlling Output Length in Neural Encoder-Decoders. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1328-1338.

Kingma, D. P. & Ba, J. (2017, January, 29). Adam: A Method for Stochastic Optimization [arXiv e-prints]. Consulted at http://arxiv.org/abs/1412.6980.

Kingma, D. P. & Welling, M. (2014). Auto-Encoding Variational Bayes. *Proceedings of the 2014 International Conference on Learning Representations (ICLR)*. Consulted at http://arxiv.org/abs/1312.6114.

Klein, G., Kim, Y., Deng, Y., Senellart, J. & Rush, A. (2017, July). OpenNMT: Open-Source Toolkit for Neural Machine Translation. *Proceedings of ACL 2017, System Demonstrations*, pp. 67–72. Consulted at https://www.aclweb.org/anthology/P17-4012.

Knight, K. & Marcu, D. (2002). Summarization Beyond Sentence Extraction: A Probabilistic Approach to Sentence Compression. *Artificial Intelligence*, 139(1), 91–107. doi: 10.1016/S0004-3702(02)00222-9.

Krause, B., Lu, L., Murray, I. & Renals, S. (2016, May, 6). Multiplicative LSTM for sequence modelling [arXiv e-prints]. Consulted at http://arxiv.org/abs/1609.07959.

Kryściński, W., Keskar, N. S., McCann, B., Xiong, C. & Socher, R. (2019). Neural Text Summarization: A Critical Evaluation. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 540-551. doi: 10.18653/v1/D19-1051.

Lawton, P. & Jankowski, T. (2009). *Investment Performance Measurement: Evaluating and Presenting Results*. Wiley.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V. & Zettlemoyer, L. [version 1]. (2019, October, 29). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension [arXiv e-prints]. Consulted at http://arxiv.org/abs/1910.13461.

Lin, C.-Y. (2004, July). ROUGE: A Package for Automatic Evaluation of Summaries. *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pp. 71-81.

Lin, C.-Y. & Hovy, E. (2003, May). Automatic Evaluation of Summaries Using N-gram Co-occurrence Statistics. *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 71-78.

Liu, P. J., Chung, Y.-A. & Ren, J. [version 1]. (2019a). SummAE: Zero-Shot Abstractive Text Summarization using Length-Agnostic Auto-Encoders. Consulted at http://arxiv.org/abs/1910.00998.

Liu, Y. & Lapata, M. (2019, November). Text Summarization with Pretrained Encoders. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3721–3731. doi: 10.18653/v1/D19-1387.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L. & Stoyanov, V. (2019b, July, 26). RoBERTa: A Robustly Optimized BERT Pretraining Approach [arXiv e-prints]. Consulted at https://arxiv.org/abs/1907.11692.

Liu, Y., Luo, Z. & Zhu, K. (2018, October). Controlling Length in Abstractive Summarization Using a Convolutional Neural Network. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4110–4119. doi: 10.18653/v1/D18-1444.

Maddison, C. J., Mnih, A. & Teh, Y. W. (2016, November, 02). The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables [arXiv e-prints]. Consulted at http://arxiv.org/abs/1611.00712.

Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I. & Frey, B. [version 2]. (2016, May, 25). Adversarial Autoencoders [arXiv e-prints]. Consulted at http://arxiv.org/abs/1511.05644.

Makino, T., Iwakura, T., Takamura, H. & Okumura, M. (2019). Global Optimization under Length Constraint for Neural Text Summarization. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 1039–1048. doi: 10.18653/v1/P19-1099.

Manning, C., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. & McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55–60. doi: 10.3115/v1/P14-5010.

Manning, C. D., Raghavan, P. & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press. Consulted at https://dl.acm.org/doi/book/10.5555/1394399.

Marcotte, E. (2011). *Responsive Web Design*. A Book Apart.

Marcu, D. (1997). From discourse structures to text summaries. *Proceedings of the ACL Workshop on Intelligent Scalable Text Summarization*, pp. 82–88. Consulted at https://www.aclweb.org/anthology/W97-0713.

McAuley, J., Targett, C., Shi, Q. & van den Hengel, A. (2015, August). Image-Based Recommendations on Styles and Substitutes. *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 43-52.

Mellish, C. S. & Dale, R. (1998). Evaluation in the context of natural language generation. *Computer Speech & Language*, 12(4), 349–373. doi: 10.1006/csla.1998.0106.

Mihalcea, R. (2004). Graph-based Ranking Algorithms for Sentence Extraction, Applied to Text Summarization. *The Companion Volume to the Proceedings of the 42nd Annual Meeting of the ACL*, pp. 170–173. Consulted at https://www.aclweb.org/anthology/P04-3020.

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013, May). Efficient Estimation of Word Representations in Vector Space. *ICLR Workshop Papers*. Consulted at http://arxiv.org/abs/1301.3781.

Nagel, S. (2016). *CC-News*. https://commoncrawl.org/2016/10/news-dataset-available/.

Nallapati, R., Zhou, B., Santos, C. d., Gulcehre, C. & Xiang, B. (2016, August). Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond. *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 280-290.

Nallapati, R., Zhai, F. & Zhou, B. (2017). SummaRuNNer: A Recurrent Neural Network Based Sequence Model for Extractive Summarization of Documents. *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pp. 3075–3081. Consulted at https://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14636/14080.

Narayan, S., Cohen, S. B. & Lapata, M. (2018). Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1797–1807. doi: 10.18653/v1/D18-1206.

Nayeem, M. T., Fuad, T. A. & Chali, Y. (2018). Abstractive Unsupervised Multi-Document Summarization using Paraphrastic Sentence Fusion. *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 1191–1204. Consulted at https://www.aclweb.org/anthology/C18-1102.

Nenkova, A. (2005, January). Automatic Text Summarization of Newswire: Lessons Learned from the Document Understanding Conference. *Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, pp. 1436-1441.

Nenkova, A. & McKeown, K. (2011). Automatic Summarization. *Foundations and Trends® in Information Retrieval*, 5(2-3), 103-233. doi: 10.1561/1500000015.

Nenkova, A. & Vanderwende, L. (2005, January). The Impact of Frequency on Summarization. *Microsoft Research, MSR-TR-2005*.

Ng, J.-P. & Abrecht, V. (2015, September). Better Summarization Evaluation with Word Embeddings for ROUGE. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1925–1930. doi: 10.18653/v1/D15-1222.

Over, P. & Yen, J. (2003). An Introduction to DUC 2003: Intrinsic Evaluation of Generic News Text Summarization Systems [Web Download. Document Understanding Conference]. Consulted at http://www-nlpir.nist.gov/projects/duc/pubs/2003slides/duc2003intro.pdf.

Owczarzak, K., Conroy, J. M., Dang, H. T. & Nenkova, A. (2012, June). An Assessment of the Accuracy of Automatic Evaluation in Summarization. *Proceedings of Workshop on Evaluation Metrics and System Comparison for Automatic Summarization*, pp. 1-9.

Parker, R., Graff, D., Kong, J., Chen, K. & Maeda, K. (2011, June, 17). English Gigaword Fifth Edition LDC2011T07 [Web Download. Linguistic Data Consortium]. Consulted at https://catalog.ldc.upenn.edu/LDC2011T07.

Parveen, D., Ramsl, H.-M. & Strube, M. (2015). Topical Coherence for Graph-based Extractive Summarization. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1949–1954. doi: 10.18653/v1/D15-1226.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. & Garnett, R. (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc. Consulted at http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Paulus, R., Xiong, C. & Socher, R. (2018, February). A Deep Reinforced Model for Abstractive Summarization. *Proceedings of the 6th International Conference on Learning Representations (ICLR)*. Consulted at https://openreview.net/forum?id=HkAClQgA-.

Pennington, J., Socher, R. & Manning, C. (2014, February). Glove: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543. doi: 10.3115/v1/D14-1162.

Peters, M., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. & Zettlemoyer, L. (2018, June). Deep Contextualized Word Representations. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1, 2227–2237. doi: 10.18653/v1/N18-1202.

Radford, A., Narasimhan, K., Salimans, T. & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training. Consulted at https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. & Sutskever, I. (2019, February, 14). Language Models are Unsupervised Multitask Learners [Web Download. OpenAI]. Consulted at https://openai.com/blog/better-language-models/.

Ranzato, M. A., Poultney, C., Chopra, S. & Cun, Y. L. (2007). Efficient Learning of Sparse Representations with an Energy-Based Model. In Schölkopf, B., Platt, J. C. & Hoffman, T. (Eds.), *Advances in Neural Information Processing Systems 19* (pp. 1137-1144). Vancouver, Canada: MIT Press.

Reiter, E. (2007, June). An Architecture for Data-to-text Systems. *Proceedings of the Eleventh European Workshop on Natural Language Generation*, pp. 97–104. Consulted at http://dl.acm.org/citation.cfm?id=1610163.1610180.

Reiter, E. & Dale, R. (2000). *Building natural language generation systems*. Cambridge University Press.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386–408. doi: 10.1037/h0042519.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088), 533–536. doi: 10.1038/323533a0.

Rush, A. M., Chopra, S. & Weston, J. (2015, September). A Neural Attention Model for Abstractive Sentence Summarization. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 379-389. doi: 10.18653/v1/D15-1044.

Sandhaus, E. (2008, October, 17). The New York Times Annotated Corpus LDC2008T19 [Web Download. Linguistic Data Consortium]. Consulted at https://catalog.ldc.upenn.edu/LDC2008T19.

Schumann, R. [version 2]. (2018). Unsupervised Abstractive Sentence Summarization using Length Controlled Variational Autoencoder. Consulted at http://arxiv.org/abs/1809.05233.

Schuster, M. & Paliwal, K. K. (2010). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11), 2673-2681.

See, A., Liu, P. J. & Manning, C. D. (2016, July). Get To The Point: Summarization with Pointer-Generator Networks. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1073-1083.

ShafieiBavani, E., Ebrahimi, M., Wong, R. & Chen, F. (2018, October). A Graph-theoretic Summary Evaluation for ROUGE. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 762–767. doi: 10.18653/v1/D18-1085.

Sutskever, I., Vinyals, O. & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D. & Weinberger, K. Q. (Eds.), *Advances in Neural Information Processing Systems 27* (pp. 3104-3112). Montréal, Canada: Curran Associates, Inc.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. & Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826. doi: 10.1109/CVPR.2016.308.

Takase, S. & Okazaki, N. (2019, June). Positional Encoding to Control Output Sequence Length. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 3999–4004. doi: 10.18653/v1/N19-1401.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. & Polosukhin, I. (2017). Attention is All you Need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S. & Garnett, R. (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 5998-6008). Long Beach, California: Curran Associates, Inc.

Vincent, P., Larochelle, H., Bengio, Y. & Manzagol, P.-A. (2008). Extracting and Composing Robust Features with Denoising Autoencoders. *Proceedings of the 25th International Conference on Machine Learning*, pp. 1096–1103. doi: 10.1145/1390156.1390294.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. & Manzagol, P.-A. (2010). Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *The Journal of Machine Learning Research*, 11, 3371-3408.

Wang, W., Pan, S. J., Dahlmeier, D. & Xiao, X. (2017). Coupled multi-layer attentions for co-extraction of aspect and opinion terms. *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pp. 3316–3322. Consulted at https://www.aaai.org/Conferences/AAAI/2017/PreliminaryPapers/15-Wang-W-14441.pdf.

Wang, Y. & Lee, H.-Y. (2018). Learning to Encode Text as Human-Readable Summaries using Generative Adversarial Networks. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 4187–4195. doi: 10.18653/v1/D18-1451.

West, P., Holtzman, A., Buys, J. & Choi, Y. (2019). BottleSum: Unsupervised and Self-supervised Sentence Summarization using the Information Bottleneck Principle. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3743–3752. doi: 10.18653/v1/D19-1389.

Williams, R. J. & Zipser, D. (1989). A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, 1(2), 270-280.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M. & Dean, J. (2016, September, 26). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation [arXiv e-prints]. Consulted at http://arxiv.org/abs/1609.08144.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A. & Hovy, E. (2016). Hierarchical Attention Networks for Document Classification. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 1480–1489. doi: 10.18653/v1/N16-1174.

Yang, Z., Zhu, C., Gmyr, R., Zeng, M., Huang, X. & Darve, E. (2020, January, 5). TED: A Pretrained Unsupervised Summarization Model with Theme Modeling and Denoising [arXiv e-prints]. Consulted at http://arxiv.org/abs/2001.00725.

Yin, W. & Pei, Y. (2015). Optimizing Sentence Modeling and Selection for Document Summarization. *Proceedings of the 24th International Conference on Artificial Intelligence*, pp. 1383–1389. Consulted at http://dl.acm.org/citation.cfm?id=2832415.2832442.

Zhang, J., Zhao, Y., Saleh, M. & Liu, P. J. (2019, December, 18). PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization [arXiv e-prints]. Consulted at http://arxiv.org/abs/1912.08777.

Zheng, H. & Lapata, M. (2019). Sentence Centrality Revisited for Unsupervised Summarization. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 6236–6247. doi: 10.18653/v1/P19-1628.

Zhou, J. & Rush, A. (2019). Simple Unsupervised Summarization by Contextual Matching. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 5101–5106. doi: 10.18653/v1/P19-1503.

Zhou, L., Lin, C.-Y., Munteanu, D. S. & Hovy, E. (2006, June). ParaEval: Using Paraphrases to Evaluate Summaries Automatically. *Proceedings of the Human Language Technology Conference of the NAAC*, pp. 447–454. Consulted at https://www.aclweb.org/anthology/N06-1057.

Zhu, J.-Y., Park, T., Isola, P. & Efros, A. A. (2017). Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2242–2251. doi: 10.1109/ICCV.2017.244.