# Simplifying Complexity: A Journey through Text Summarization

Lok Yee Joey Cheung

**Abstract**

Text summarization is a crucial task in Natural Language Processing which addresses the pressing need for condensing a lengthy text into a short, concise summary while capturing relevant information. This tutorial paper aims to provide an in-depth exploration of extractive and abstractive text summarization techniques within the domain of deep learning, with a deeper exploration of the latter. Specifically, Recurrent Neural Network-based Sequence-to-Sequence model (Seq2Seq) and recent advancements will be our focus. The paper will also discuss evaluation metrics and the analysis of the strengths and limitations of the techniques. By the end, readers will gain a comprehensive understanding of text summarization techniques and insights into practical implications.

## 1 Introduction

In this digital era, we are exposed to vast amounts of digital content. The ability to distill and extract key information efficiently is essential, particularly in professions like research, data analysis, journalism and more. Summarization also improves the accessibility of complex text to individuals with limited time or cognitive resources. Numerous automatic text summarization platforms have emerged, and delving into their theories and mechanisms behind is intriguing. This tutorial paper explores key concepts of extractive and abstractive text summarization techniques. We will commence by understanding basic extractive summarization methods, followed by a more extensive discussion on abstractive summarization. Throughout this tutorial, we will elucidate the underlying principles of these summarization techniques, training procedures, and evaluation methodologies. Furthermore, we will discuss their applications, strengths and limitations. Text summarization tackles the challenge of information overload, fosters knowledge extraction, improves accessibility, and boosts efficiency in various applications, making it a significant area of research and development in Natural Language Processing.

## 2 Extractive Text Summarization

Extractive text summarization involves extracting relevant sentences directly from the original text to form a concise summary (Saxena and El-Haj, 2023). This approach does not rephrase and restructure the content. For instance, frequency-based approaches rely on statistical measures namely word frequency or sentence position to calculate the importance of words and phrases (Saxena and El-Haj, 2023). Examples include TF-IDF, SumBasic and SumFocus (Saxena and El-Haj, 2023).

### 2.1 TF-IDF

Term Frequency - Inverse Document Frequency (TF-IDF) measures the importance of a word within a document compared to a broader set of documents (Ramos et al., 2003). Sentences are sorted by TF-IDF score and those with the highest scores will be selected as the outputs (Allahyari et al., 2017).

$$w_d = f_{w,d} * log(|D|/f_{w,D})$$

where $f_{w,d}$ denotes the frequency of word $w$ in document $d$, $|D|$ represents the total number of documents and $f_{w,D}$ is the number of documents containing word $w$.

**Exercise 1:**

Practice extractive text summarization using TD-IDF.

**Answer:**

Please refer to Appendix A for sample code.

More extractive summarization approaches have been explored in recent decades. Since we will be focusing on deep learning in abstractive summarization, please feel free to explore the extractive methodologies by yourself. Reference can be found in Giarelis, Mastrokostas, and Karacapilidis, 2023; Mihalcea, 2004; Moratanch and Chitrakala, 2017.

## 3 Abstractive Text Summarization

Abstractive summarization generates new sentences to convey the main idea of the original text by paraphrasing and synthesizing the content (Cohan et al., 2018). They frequently utilize deep learning architectures like Recurrent Neural Networks (RNNs), Long Short-Term Memory Networks (LSTMs), or Transformer models, namely BERT, GPT, or T5.

### 3.1 RNN Encoder-Decoder

Most of the text summarization algorithms are built upon Machine Translation attentional RNN encoder-decoder model (Hu, Chen, and Zhu, 2015; Nallapati et al., 2016). Let us recap some of the fundamental concepts. When an input sequence $x = (x_1, ..., x_{Tx})$ is passed into a bi-directional encoder. The encoder processes the input and updates the hidden states at time t, $h_t^e$ (Bahdanau, Cho, and Bengio, 2014):

$$h_t^e = f^e(x_t, h_{t-1})$$

where $f$ and $q$ are some non-linear functions like LSTM or GRU etc. The bi-directional encoder consists of forward and backward RNNs (Bahdanau, Cho, and Bengio, 2014). The former reads input in order and produces $(\overrightarrow{h_1^e}, \ldots, \overrightarrow{h_{\text{Tx}}^e})$ while the latter reads them in reverse order and calculates $(\overleftarrow{h_1^e}, \ldots, \overleftarrow{h_{\text{Tx}}^e})$. Therefore, the hidden states at time t, $h_t^e$ equals to:

$$h_t^e = [\overrightarrow{h_t^e}, \overleftarrow{h_t^e}]$$

which attends to and captures the information of the preceding and following words (Bahdanau, Cho, and Bengio, 2014). The context vector $c_i$ is the weighted sum of the sequence of hidden states (Bahdanau, Cho, and Bengio, 2014):

$$c_i = \sum_{j=1}^{Tx} \alpha_{ij} h_j^e$$

where $\alpha_{ij}$ is the weight (i.e. importance) of each hidden state, $h_j^e$ concerning the preceding $h_{t-1}^d$ in determining the subsequent $h_t^d$ and producing $y_i$. This is called the attention mechanism (Bahdanau, Cho, and Bengio, 2014). The decoder is trained to predict the next word using current hidden state, context vector and all predicted words $\{y_1, ..., y_{t'-1}\}$:

$$h_t^d = f^d(y_{t-1}, h_{t-1}^d, c)$$

and

$$y_t \sim p(y_t|y_1, ..., y_{t-1}, c) = g(y_{t-1}, h_t^d, c)$$

where $y = (y_1, ..., y_{Ty})$ is the output sequence and $g$ stands for some non-linear functions for generating the probability of $y_t$. Nallapati et al., 2016 expanded upon the above by enriching input features with embedding vectors representing parts of speech and named-entity tags etc. Additionally, they integrated a pointer network to copy unseen words from the source text to address out-of-vocabulary word problems.

**Questions:**

(a) Based on your current knowledge, what are the similarities and differences between Machine Translation and Text Summarization?

(b)What type of sequence modeling is employed in text summarization?

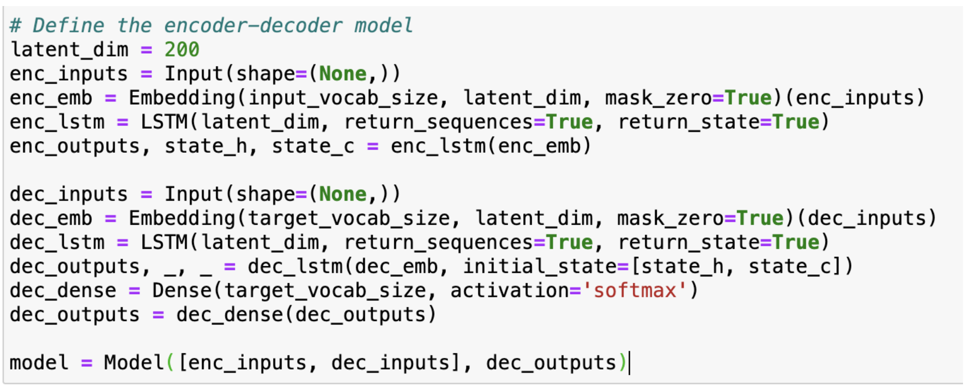
**Answer:**

(a) Both of them are natural language processing tasks that employ Seq2Seq modeling architectures, utilizing many-to-many sequence modeling. As for differences, summarization's target output tends to be concise and is not affected by the length of the input document. A significant difference is that summarization aims to effectively compress the original document in a lossy manner while retaining its key information. In contrast, machine translation preserves all details in a loss-less way (Nallapati et al., 2016).

(b)As mentioned above, it is a many-to-many sequence modeling problem, where the model processes multiple input tokens to generate multiple output tokens. This enables flexible mapping between input and output sequences, making it suitable for tasks like text summarization, where the length of the input and output may differ. By employing a many-to-many sequence modeling approach, the model can effectively capture the semantic structure and essential information of the input text and generate concise summaries (Bahdanau, Cho, and Bengio, 2014).

## 3.2 RNN Seq2Seq Model Example Code

We will now comprehend the model architecture by studying the following code snippets. It uses LSTM as the backbone of the summarizer and takes reference from (Boynton, 2023) and (Sriramulugari, 2024).

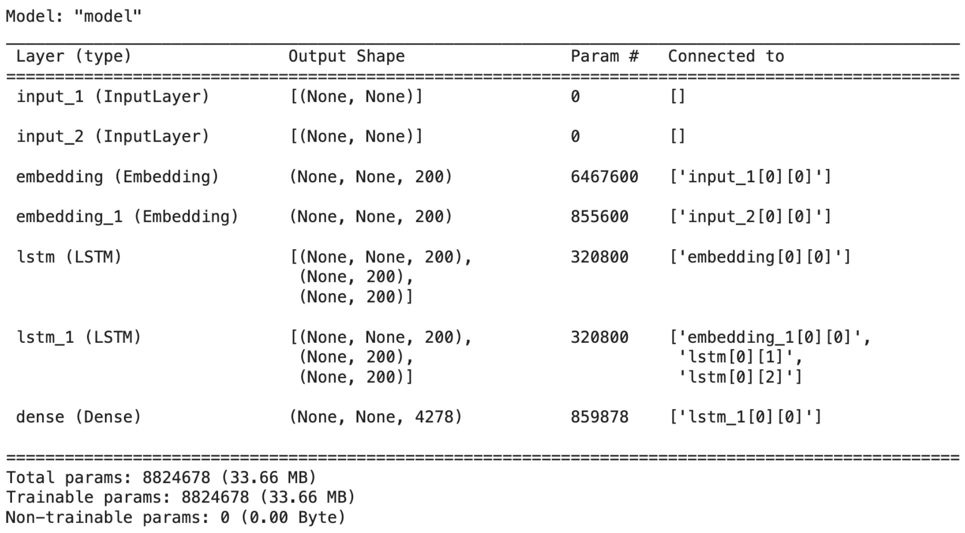


```python
# Define the encoder-decoder model
latent_dim = 200
enc_inputs = Input(shape=(None,))
enc_emb = Embedding(input_vocab_size, latent_dim, mask_zero=True)(enc_inputs)
enc_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
enc_outputs, state_h, state_c = enc_lstm(enc_emb)

dec_inputs = Input(shape=(None,))
dec_emb = Embedding(target_vocab_size, latent_dim, mask_zero=True)(dec_inputs)
dec_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
dec_outputs, _, _ = dec_lstm(dec_emb, initial_state=[state_h, state_c])
dec_dense = Dense(target_vocab_size, activation='softmax')
dec_outputs = dec_dense(dec_outputs)

model = Model([enc_inputs, dec_inputs], dec_outputs)
```

Figure 1: Example Encoder-Decoder Model



```
Model: "model"
_____________________________________________________________________________________________________
 Layer (type)                 Output Shape                 Param #   Connected to
=====================================================================================================
 input_1 (InputLayer)         [(None, None)]               0         []

 input_2 (InputLayer)         [(None, None)]               0         []

 embedding (Embedding)        (None, None, 200)            6467600   ['input_1[0][0]']

 embedding_1 (Embedding)      (None, None, 200)            855600    ['input_2[0][0]']

 lstm (LSTM)                  [(None, None, 200),          320800    ['embedding[0][0]']
                               (None, 200),
                               (None, 200)]

 lstm_1 (LSTM)                [(None, None, 200),          320800    ['embedding_1[0][0]',
                               (None, 200),                           'lstm[0][1]',
                               (None, 200)]                           'lstm[0][2]']

 dense (Dense)                (None, None, 4278)           859878    ['lstm_1[0][0]']

=====================================================================================================
Total params: 8824678 (33.66 MB)
Trainable params: 8824678 (33.66 MB)
Non-trainable params: 0 (0.00 Byte)
```

Figure 2: Example Model Summary

In Figure 1, the first layer of the model is the input layer which takes variable lengths of input text. In the second layer, the embedding layer takes the vocabulary size of the input tokenizer ('input_vocab_size') and the dimension of the dense vector representation for words ('latent_dim'). The LSTM layer processes the embedded input sequences and captures sequential information. Its hidden states' size is also determined by 'latent_dim' which is set to 200. A higher latent dimension allows the model to learn more complex relationships in the data while increasing the computational complexity. 'enc_outputs', 'state_h' and 'state_c' represent the sequences of hidden states, the final hidden state and the context vector of the encoder LSTM layer.

The decoder, similarly, receives variable-length target sequences and utilizes an embedding layer and an LSTM layer to generate output sequences, conditioned on the context vector produced by the encoder. The decoder's output is then passed through a dense layer with SoftMax activation to predict the probability distribution of each token in the target vocabulary. The model summary in Figure 2 provides a concise overview of the model architecture and detailed connections between layers.

**Exercise 2:**

Practice abstractive text summarization using RNN, define an encoder-decoder model and train it for summarization task.

**Answer:**

Please refer to Appendix A for full code.

RNN encounters challenges during training when gradients can vanish or explode. This leads to failure in capturing long-term dependencies. Besides, as deep learning models grow in size and complexity, traditional attention mechanisms can become computationally and memory-intensive. Transformers address this challenge by introducing self-attention mechanisms, enabling parallel processing of the entire input sequence and efficient capture of contextual information, thereby revolutionizing the approach to sequence-to-sequence tasks.

## 3.3 Transformer

Transformer model uses non-recurrent architecture and relies heavily on self-attention mechanism (Vaswani et al., 2017). Unlike RNN, transformer is highly parallelizable since all words in the input sequence are processed simultaneously. It can be trained more effectively on longer input sequence summarization using GPUs and TPUs. This not only improves the performance of large-scale models but also enhances their computational efficiency (Vaswani et al., 2017).

It consists of two major parts: encoder and decoder. The vector embeddings of the input sequence will pass through a series of encoding and decoding layers. One of the key components is the self-attention mechanism (Vaswani et al., 2017). It allows each token to attend to all other tokens in the sequence, capturing contextual information between words (Vaswani et al., 2017). Attention is computed as follows:

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

where $Q$ vectors are the current word or token being processed, $K$ vectors and $V$vectors are all words or tokens in the input text, and $d_k$ is the dimensions of these vectors. SoftMax function is applied to obtain the weight of the values. Transformer also performs multi-head attention which allows parallelization of self-attention and captures more diverse relationships in the data (Vaswani et al., 2017).

Moreover, each layer of both the encoder and decoder exhibits a fully-connected feed-forward network, $FFN$. This network operates on each token within a sequence independently and uniformly (Vaswani et al., 2017). It comprises two linear transformations with a ReLU activation in between:

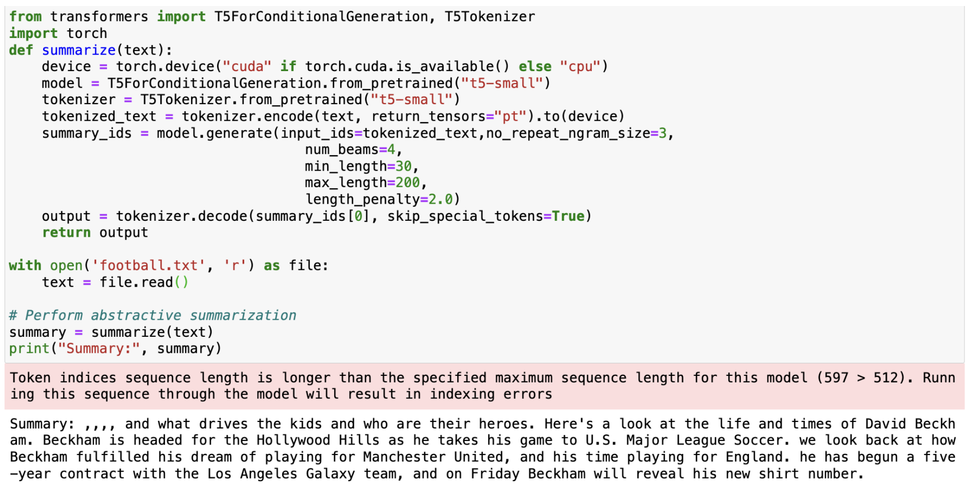
$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

The output of the attention mechanism will go through the FFN and the resulting contextualized representations are passed to the decoder to generate the output sequence (Vaswani et al., 2017). The model uses linear transformation and SoftMax function to compute the probabilities of each token and select the token with the highest probability as the predicted output (Vaswani et al., 2017). Please refer to Appendix B for the full Transformer architecture (Vaswani et al., 2017).

## 3.4 Transformer Example Code

We will adopt T5 (text-to-text transfer transformer), one of the renowned Transformer models that convert text-based NLP tasks into text-to-text format (Guo et al., 2021). To implement the T5 model, we will use the Hugging Face 'Transformers' library. You can find the 'T5-small' model ('T5ForConditionalGeneration') and tokenizer ('T5Tokenizer') on the Hugging Face Model Hub (`https://huggingface.co/t5-small`). Please feel free to use CNN/DailyMail dataset for your task which consists vast amount of news articles written by CNN's and Daily Mail's journalists (community, n.d.).

The code in Figure 3 demonstrates the process of performing abstractive summarization using T5. After performing tokenization, we generate the model by setting parameters. 'num_beams' denotes the number of sequences being considered during beam search. 'min_length' and 'max_length' set the minimum and maximum length of generated summary respectively. 'length_penalty' is an exponential penalty allocated to length during beam search.



```
from transformers import T5ForConditionalGeneration, T5Tokenizer
import torch
def summarize(text):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = T5ForConditionalGeneration.from_pretrained("t5-small")
    tokenizer = T5Tokenizer.from_pretrained("t5-small")
    tokenized_text = tokenizer.encode(text, return_tensors="pt").to(device)
    summary_ids = model.generate(input_ids=tokenized_text,no_repeat_ngram_size=3,
                                num_beams=4,
                                min_length=30,
                                max_length=200,
                                length_penalty=2.0)
    output = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
    return output

with open('football.txt', 'r') as file:
    text = file.read()

# Perform abstractive summarization
summary = summarize(text)
print("Summary:", summary)
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model (597 > 512). Runn
ing this sequence through the model will result in indexing errors
```

```
Summary: ,,,, and what drives the kids and who are their heroes. Here's a look at the life and times of David Beckh
am. Beckham is headed for the Hollywood Hills as he takes his game to U.S. Major League Soccer. we look back at how
Beckham fulfilled his dream of playing for Manchester United, and his time playing for England. he has begun a five
-year contract with the Los Angeles Galaxy team, and on Friday Beckham will reveal his new shirt number.
```

Figure 3: Perform T5 for abstractive summarization

**Exercise 3:**

(a) Practice the above T5 model implementation with different parameters and spot the impacts.

(b) During the implementation, you will encounter an error indicating that the input text you are trying to summarize is longer than the maximum sequence length that the T5 model can handle, which is 512 tokens for the 't5small' model. Try to solve it by segmenting your input into smaller chunks and summarize each chunk separately. Finally, concatenate the summaries.

(c) Supplementary: You can also try to implement LongT5 (`https://huggingface.co/docs/transformers/model_doc/longt5`), which is a extension of the original T5 model to handle longer inputs (Guo et al., 2021). What differences do you observe in the results?

**Answer:**

Please see the Appendix B.

## 4 Evaluation Metrics

ROUGE score compares the quality of the generated summaries with the human-annotated reference summaries (Rouge, 2004). This automated evaluation involves counting overlapping units namely word sequences, n-grams, and word pairs among the summaries (Rouge, 2004). I will introduce ROUGE-N and ROUGE-L. ROUGE-N measures the co-occurring n-grams between the candidate and the reference summary:

$$ROUGE_N = \frac{\sum_{S \in ReferenceSummaries} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in ReferenceSummaries} \sum_{gram_n \in S} Count(gram_n)}$$

where $gram_n$ represents the length of the n-gram, $Count_{match}(gram_n)$ denotes the highest co-occurrence of n-grams in the two summaries. Moreover, ROUGE-L measures the longest common subsequence (LCS) between both summaries:

$$Recall_{lcs} = \frac{LCS(X,Y)}{m}$$

$$Precision_{lcs} = \frac{LCS(X,Y)}{n}$$

$$F_{lcs} = \frac{(1+\beta^2) * Recall_{lcs} * Precision_{lcs}}{Recall_{lcs} + \beta^2 * Precision_{lcs}}$$

**Exercise 4:**

Try to evaluate your previously generated summaries with Rouge by importing Rouge metrics in Python.

**Answer:**

See Appendix C for a detailed code demo.

## 5 Strengths and Limitations

The following table compares the extractive and abstractive methods. How should we choose the appropriate approach? While extractive summarization offers the advantage of preserving factual accuracy through extracting important sentences, it tends to suffer from the lack of coherence between sentences (Giarelis, Mastrokostas, and Karacapilidis, 2023). It may be more suitable for news aggregation or document summarization that requires a high degree of accuracy.

On the other hand, abstractive summarization addresses the above limitation by generating summaries with higher readability. It consolidates the semantic meaning in the text and paraphrases the sentences (Cohan et al., 2018). Tasks that require opinion summarization such as recommendation system (Condori and Pardo, 2017). More applications like automatic text generation, and educational and accessibility tools may also prefer this approach. However, it is often computationally intensive. The choice between the two approaches depends on the requirements and nature of tasks, the desired characteristics of the summaries, the resources available and more.

Table 1: Table of Comparison

| | **Extractive** | **Abstractive** |
|---|---|---|
| Approach | Identify and select important sentences | Understand overall context and generate summaries |
| Accuracy | Higher factual accuracy due to direct extraction of sentences from text | Prone to factual errors |
| Readability | May lack coherence between sentences | Produce more readable and concise summaries by rephrasing complex sentences |
| Complexity | Generally simpler and faster to implement | More computational complex due to the use of deep learning models |

## Appendix A.

N.B. Replace the sample article that I provided with any article you like for summarization tasks.

**Code demo of performing extractive summarization using TF-IDF (Exercise 1):**

```
import nltk
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np

def tfidf_summarize(text, num_sentences):
    # Tokenize the text
    sentences = nltk.sent_tokenize(text)
    vect = TfidfVectorizer(stop_words='english')
    tfidf_matrix = vect.fit_transform(sentences)

    # Calculate the average TF-IDF score for each sentence
    tfidf_sentence = np.mean(tfidf_matrix.toarray(), axis=1)

    # Get the top-ranked sentences
    top_idx = tfidf_sentence.argsort()[-num_sentences:][::-1]
    top_idx.sort()
    summary = ' '.join([sentences[i] for i in top_idx])

    return summary

# Perform extractive text summarisation
with open('football.txt', 'r') as file:
    text = file.read()

print(tfidf_summarize(text,3))
```

**Code demo of performing abstractive summarization using RNN (Exercise 2):**

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Model
from keras.layers import Input, LSTM, Dense, Embedding
from keras.models import load_model

def tokenize(train,val):
  tokenizer = Tokenizer()
  tokenizer.fit_on_texts(train)
  tr_seq = tokenizer.texts_to_sequences(train)
  val_seq = tokenizer.texts_to_sequences(val)
  max_len = max([len(seq) for seq in tr_seq])
  vocab_size = len(tokenizer.word_index) + 1

  return tr_seq, val_seq, max_len, vocab_size

def summary(txt):
    input = input_tok.texts_to_sequences([txt])
    input = pad_sequences(input, maxlen=max_input_len, padding='post')
    target = np.zeros((1, max_target_len), dtype='int')
    target[0,0] = target_tok.word_index.get('<sos>', 0)
    summary = ''

    stop = False
    while not stop:
        outputs = model.predict([input, target])
        pred_idx = np.argmax(outputs[0, 0, :])
        pred_word = None
        for word, idx in target_tok.word_index.items():
            if idx == pred_idx:
                summary += ' {}'.format(word)
                pred_word = word
        if pred_word == '<eos>' or len(summary.split()) > max_target_len:
            stop = True
        target[0, 0] = pred_idx

    return summary

data = pd.read_csv("cnn_1000.csv")
data["text"].fillna("", inplace=True)
data["summary"].fillna("", inplace=True)

x_tr, x_val, y_tr, y_val = train_test_split(
    np.array(data["text"]),np.array(data["summary"]),
    test_size=0.1,random_state=0,shuffle=True)

# Tokenize input texts and target texts
x_tr_seq, x_val_seq, max_input_len,input_vocab_size = tokenize(x_tr, x_val)
y_tr_seq, y_val_seq, max_target_len,target_vocab_size= tokenize(y_tr, y_val)

# Pad sequences to make input and target uniform in length
enc_input_tr = pad_sequences(x_tr_seq, maxlen=max_input_len, padding='post')
enc_input_val = pad_sequences(x_val_seq, maxlen=max_input_len, padding='post')
dec_input_tr = pad_sequences(y_tr_seq, maxlen=max_target_len, padding='post')
dec_input_val = pad_sequences(y_val_seq, maxlen=max_target_len, padding='post')

# Shift target sequences by one position
dec_target_tr = np.zeros_like(dec_input_tr)
dec_target_tr[:, 0:-1] = dec_input_tr[:, 1:]
dec_target_tr[:, -1] = 0
dec_target_val = np.zeros_like(dec_input_val)
dec_target_val[:, 0:-1] = dec_input_val[:, 1:]
dec_target_val[:, -1] = 0

# Define the encoder-decoder model
latent_dim = 200
enc_inputs = Input(shape=(None,))
enc_emb = Embedding(input_vocab_size, latent_dim, mask_zero=True)(enc_inputs)
enc_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
enc_outputs, state_h, state_c = enc_lstm(enc_emb)

dec_inputs = Input(shape=(None,))
dec_emb = Embedding(target_vocab_size, latent_dim, mask_zero=True)(dec_inputs)
dec_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
dec_outputs, _, _ = dec_lstm(dec_emb, initial_state=[state_h, state_c])
dec_dense = Dense(target_vocab_size, activation='softmax')
dec_outputs = dec_dense(dec_outputs)

model = Model([enc_inputs, dec_inputs], dec_outputs)
model.summary()

# Compile and train the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')
model.fit([enc_input_tr, dec_input_tr], dec_target_tr,
          validation_data=([enc_input_val, dec_input_val], dec_target_val),
          batch_size=64, epochs=50)

# Save the model optionally
#model.save("encoder_decoder_model.h5")
#model = load_model("encoder_decoder_model.h5")

input_tok = Tokenizer()
input_tok.fit_on_texts(x_val)
target_tok = Tokenizer()
target_tok.fit_on_texts(y_val)
for i in range(len(x_val)):
  print("Article:", x_val[i])
  print("Reference summary:", y_val[i])
  print("Candidate summary:",summary(x_val[i]))
```

## Appendix B.

**The following figure shows the Transformer architecture:**

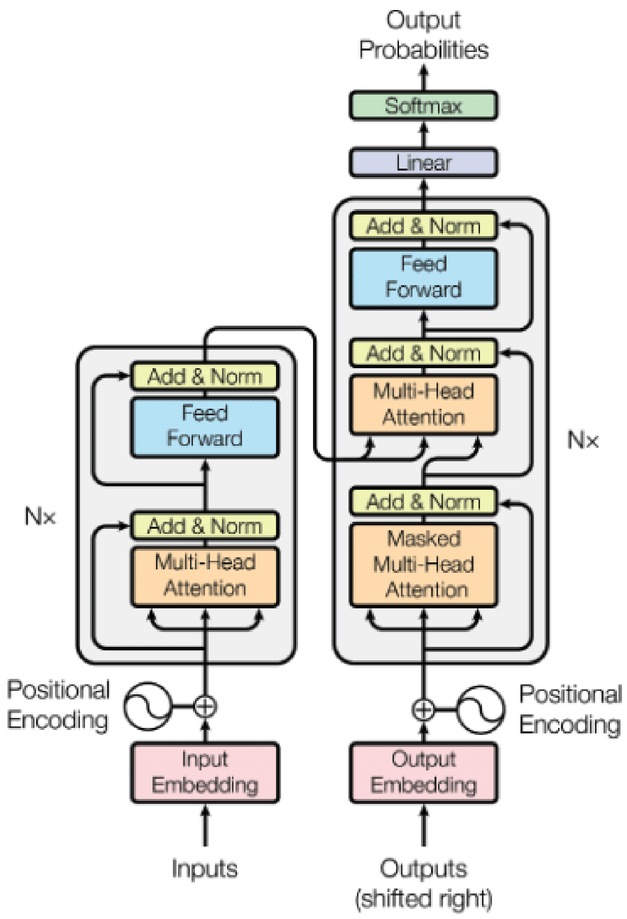


Figure 4: Transformer architecture

### Code demo of T5 model (Exercise 3a):

```
from transformers import T5ForConditionalGeneration, T5Tokenizer
import torch
def summarize(text):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = T5ForConditionalGeneration.from_pretrained("t5-small")
    tokenizer = T5Tokenizer.from_pretrained("t5-small")
    tok_txt = tokenizer.encode(text, return_tensors="pt").to(device)
    summary_ids = model.generate(input_ids=tok_txt,
                                num_beams=4,min_length=30,max_length=200,
                                length_penalty=2.0)
    output = tokenizer.decode(summary_ids[0], skip_special_tokens=True)
    return output

with open('football.txt', 'r') as file:
    text = file.read()

# Perform abstractive summarization
summary = summarize(text)
print("Summary:", summary)
```

### Code demo of segmenting the input to fit the T5 model (Exercise 3b):

```
from transformers import T5ForConditionalGeneration, T5Tokenizer
import torch

def summarizeSeg(text):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = T5ForConditionalGeneration.from_pretrained("t5-small")
    tokenizer = T5Tokenizer.from_pretrained("t5-small")

    # Split text into chunks of maximum length 512
    size = 512
    chunks = [text[i:i+size] for i in range(0, len(text), size)]

    # Summarize each chunk
    summaries = []
    for chunk in chunks:
        tokenized_text = tokenizer.encode(chunk, return_tensors="pt").to(device)
        ids = model.generate(input_ids=tokenized_text,num_beams=4,
                             max_length=200, min_length=30,
                             length_penalty=2.0)
        summary = tokenizer.decode(ids[0], skip_special_tokens=True)
        summaries.append(summary)

    # Concatenate summaries
    output = ' '.join(summaries)
    return output

with open('football.txt', 'r') as file:
    text = file.read()

summary = summarizeSeg(text)
print("Summary:", summary)
```

**Code demo of LongT5 (Exercise 3c):**

```
from transformers import AutoTokenizer, LongT5ForConditionalGeneration
import torch

def summarizeLong(text):
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    long_t5 = LongT5ForConditionalGeneration.from_pretrained(
        "Stancld/longt5-tglobal-large-16384-pubmed-3k_steps"
    )
    tokenizer = AutoTokenizer.from_pretrained(
        "Stancld/longt5-tglobal-large-16384-pubmed-3k_steps"
    )
    tok_txt = tokenizer.encode(text, return_tensors="pt").to(device)
    ids = long_t5.generate(
        tok_txt,num_beams=4,max_length=200,min_length=30,
        length_penalty=2.0
    )
    output = tokenizer.decode(ids[0], skip_special_tokens=True)
    return output

with open('football.txt', 'r') as file:
    text = file.read()

summary = summarizeLong(text)
print("Summary:", summary)
```

# Appendix C.

## Evaluation using ROUGE metrics (Exercise 4):

```
import rouge
from rouge import Rouge

# Reference summary from CNN daily mail
reference_summary = '''
```

```
Beckham has agreed to a five-year contract with Los Angeles Galaxy .
New contract took effect July 1, 2007 .
Former English captain to meet press, unveil new shirt number Friday .
CNN to look at Beckham as footballer, fashion icon and global phenomenon .'''

# Evaluate the summary
rouge = Rouge()
scores = rouge.get_scores(reference_summary, summary)
print("ROUGE score: ",scores)
```

## References

Allahyari, M. et al. (2017). “Text summarization techniques: a brief survey”. In: *arXiv preprint arXiv:1707.02268.*

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

Boynton, D. (2023). *Seq2seq news article summary-using an encoder-decoder LSTM to summarize text.* URL: `https://medium.com/@duncanboynton/seq2seq-news-article-summary-using-an-encoder-decoder-lstm-to-summarize-text-5de56fccfbf6`.

Cohan, A. et al. (2018). “A discourse-aware attention model for abstractive summarization of long documents”. In: *arXiv preprint arXiv:1804.05685.*

community, T. H. D. (n.d.). $Cnn_dailymaildatasetsathuggingface$. URL: `https://huggingface.co/datasets/cnn_dailymail`.

Condori, R. E. L. and T. A. S. Pardo (2017). “Opinion summarization methods: Comparing and extending extractive and abstractive approaches”. In: *Expert Systems with Applications* 78, pp. 124–134.

Giarelis, N., C. Mastrokostas, and N. Karacapilidis (2023). “Abstractive vs. Extractive Summarization: An Experimental Review”. In: *Applied Sciences* 13.13, p. 7620.

Guo, M. et al. (2021). “LongT5: Efficient text-to-text transformer for long sequences”. In: *arXiv preprint arXiv:2112.07916.*

Hu, B., Q. Chen, and F. Zhu (2015). “Lcsts: A large scale chinese short text summarization dataset”. In: *arXiv preprint arXiv:1506.05865.*

Mihalcea, R. (2004). “Graph-based ranking algorithms for sentence extraction, applied to text summarization”. In: *Proceedings of the ACL interactive poster and demonstration sessions*, pp. 170–173.

Moratanch, N and S Chitrakala (2017). “A survey on extractive text summarization”. In: *2017 international conference on computer, communication and signal processing (ICCCSP).* IEEE, pp. 1–6.

Nallapati, R. et al. (2016). “Abstractive text summarization using sequence-to-sequence rnns and beyond”. In: *arXiv preprint arXiv:1602.06023.*

Ramos, J. et al. (2003). “Using tf-idf to determine word relevance in document queries”. In: *Proceedings of the first instructional conference on machine learning.* Vol. 242. 1. Citeseer, pp. 29–48.

Rouge, L. C. (2004). “A package for automatic evaluation of summaries”. In: *Proceedings of Workshop on Text Summarization of ACL, Spain.* Vol. 5.

Saxena, P. and M. El-Haj (2023). “Exploring Abstractive Text Summarisation for Podcasts: A Comparative Study of BART and T5 Models”. In: *Proceedings of the 14th International Conference on Recent Advances in Natural Language Processing*, pp. 1023–1033.

Sriramulugari, S. K. (2024). *Mastering text summarization with NLP - DZone.* URL: `https://dzone.com/articles/navigating-the-complexities-of-text-summarization`.

Vaswani, A. et al. (2017). “Attention is all you need”. In: *Advances in neural information processing systems* 30.

*I give consent for this to be used as a teaching resource.*