TEXT SUMMARIZATION USING NLP

A PROJECT REPORT

submitted to Centre Of Development Of Advanced Computing, ACTS, Bengaluru
In the partial fulfilment of the requirements
for the award of the degree

DIPLOMA IN BIG DATA ANALYTICS
(PG-DBDA)

BY

Ms. Rashi Jadhav (230350125063)
Mr. Gaurav Patil (230350125023)
Mr. Utkarsh Kolhe (230350125085)

Under the guidance of

Mr. Sujit Pulujkar

DIPLOMA IN BIG DATA ANALYTICS
CENTRE OF DEVELOPMENT OF ADVANCED COMPUTING,
ACTS BENGALURU

March 2023

Acknowledgment

We feel happy in forwarding this project report as an image of sincere efforts. The successful project reflects our work, effort of my guide in giving us good information.

We would like to express our sincere gratitude to our esteemed project guide Mr. **Sujit Pulujkar** for his insightful guidance, invaluable suggestions, helpful information and practical advice which have helped us tremendously at all times. His immense knowledge, profound experience and professional expertise has enabled us to complete our project successfully.

Ms. Rashi Jadhav
Mr. Gaurav Patil
Mr. Utkarsh Kolhe

CERTIFICATE

This is to certify that **Ms. Rashi Jadhav, Mr. Gaurav Patil , Mr. Utkarsh Kolhe** has successfully submitted his/her project report to the Centre Of Development Of Advanced Computing, ACTS, Bengaluru, on **"Abstractive Text Summarization using NLP"** during the academic year 2023-2024 in the partial fulfilment towards completion of **Diploma in Big Data Analytics (PG-DBDA).**

Guide

Mr. Sujit Pulujkar

Place : Centre Of Development Of Advanced Computing, ACTS, Bengaluru.

Date :28-Aug-2023 _____

Abstract

Text summarization is a pivotal task in Natural Language Processing (NLP) aimed at condensing lengthy documents into concise and coherent summaries. This abstract presents a comprehensive overview of text summarization techniques, focusing on the advancements achieved through the integration of neural network models. The abstract discusses both extractive and abstractive summarization approaches, detailing their underlying mechanisms and methodologies. Various neural architectures, including Long Short-Term Memory (LSTM) networks and Transformer models, have revolutionized the field by enabling more effective capture of contextual information and semantic relationships within the text. Furthermore, pre-trained language models like BERT and GPT-3 have shown remarkable promise in generating coherent abstractive summaries. The abstract also touches upon the challenges faced in automatic summarization, such as content selection, fluency, and the generation of informative yet concise summaries. Through an exploration of seminal research papers and studies, this abstract offers insights into the evolution of text summarization, highlighting the potential directions for future research and applications in real-world scenarios.

Keywords Keywords related to text summarization include:

1. Text Summarization: The process of condensing a longer piece of text into a shorter version while retaining its main ideas and key points.

2. Extractive Summarization: A method of text summarization that involves selecting and extracting sentences or phrases directly from the original text to form a summary.

3. Abstractive Summarization: A method of text summarization that involves generating new sentences that capture the essence of the original text. It often involves rephrasing and paraphrasing.

4. Natural Language Processing (NLP): The field of AI that focuses on the interaction between computers and human language, including tasks like understanding, generation, and summarization.

5. Recurrent Neural Networks (RNNs): A class of neural networks designed for sequence data, often used in abstractive summarization to capture context and dependencies.

6. Long Short-Term Memory (LSTM): A type of RNN architecture that addresses the vanishing gradient problem, making it effective for learning dependencies in sequential data.

7. Attention Mechanism: A technique used in sequence-to-sequence models to focus on different parts of the input text when generating each word of the summary.

8. Transformer Architecture: A neural network architecture that utilizes self-attention mechanisms to process input data in parallel and has been the foundation for many state-of-the-art models.

Contents

Chapter 1

SOFTWARE REQUIREMENTS SPECIFICATION

1.1 Purpose

Abstractive text summarization is a natural language processing (NLP) technique used to generate concise and coherent summaries of longer texts. Unlike extractive summarization, where existing sentences or phrases are selected and combined, abstractive summarization involves generating new sentences that capture the essence of the original content while potentially rephrasing and restructuring the information. In essence, the purpose of abstractive text summarization using NLP is to distill complex or lengthy texts into coherent, informative, and concise summaries that facilitate quick understanding and decision-making.

The purpose of abstractive text summarization using NLP includes several key objectives:

Conciseness, Information Compression, Reduced Cognitive Load, Content Extraction and Fusion, Paraphrasing and Language Adaptation, Content Skimming, Social Media and Mobile Devices, Language Translation, Content Personalization, Content Indexing and Retrieval.

1.2 Model

This code appears to be a comprehensive implementation of a sequence-to-sequence (seq2seq) model for abstractive text summarization using LSTM and Bidirectional LSTM layers. It involves data preprocessing, tokenization, model architecture definition, training, and evaluation. Below, I'll provide a model description and a functional description of the code.

Model Description:

The implemented model is a seq2seq model with an encoder-decoder architecture. It's designed for abstractive text summarization, which means it generates a summary of a given input text. The model comprises the following components:

Encoder:

- The encoder takes input sequences (texts) and processes them using LSTM and Bidirectional LSTM layers.
- The input sequences are tokenized using an Embedding layer with a pretrained embedding matrix (such as GloVe embeddings).
-

Decoder:

- The decoder generates summaries based on the encoded information from the encoder.
- It uses an Embedding layer with a separate pretrained embedding matrix for the summary language.
- The decoder also uses LSTM layers to process the generated summary step by step.

Training:

- The model is trained using the Keras framework.
- The loss function used is 'sparse_categorical_crossentropy', as it's suitable for sequences with discrete values.

Hyperparameters:

- The embedding dimension is set to 300.
- The latent dimension (hidden state size of LSTM layers) is set to 240.
- The model is trained for 50 epochs with an early stopping mechanism to prevent overfitting.

Functional Model Description:

1. Data Preprocessing: The code starts by loading and preprocessing the data from CSV files. It performs various text preprocessing tasks such as expanding contractions, lowercasing, removing punctuation, numbers, and stopwords.

2**. Tokenization and Padding**: The text and summary sequences are tokenized using the Keras Tokenizer. Sequences longer than specified maximum lengths are padded. The tokenizer is also filtered to exclude rare words.

3**. Model Architecture:** The code provides three options for model architecture: one with LSTM layers, one with Bidirectional LSTM layers, and a hybrid model with Bidirectional LSTMs. Each model type is built within the TPU strategy scope.

4. **Training and Evaluation:** The chosen model is compiled and trained using training and validation data. The history of training is stored and can be visualized using plots for accuracy and loss. The training process includes early stopping and learning rate reduction.

.

1.3 Specific Requirements

The provided code appears to be a comprehensive implementation of a sequence-to-sequence (seq2seq) model for text summarization using deep learning techniques. Below are the specific requirements and components of the code:

1. Data Preparation and Preprocessing:
 - Import necessary libraries for data manipulation, visualization, and machine learning.
 - Load and preprocess two CSV files containing news headlines and their corresponding text.

2. Text Preprocessing:
 - Perform text cleaning, including expanding contractions, converting to lowercase, removing punctuation and numbers, and removing stop words.
 - Tokenize text data using the Tokenizer from TensorFlow.
 - Pad sequences to ensure uniform length for input and output sequences.

3. Word Embeddings:
 - Load pre-trained GloVe word embeddings for creating embedding matrices.
 - Generate embedding matrices for the input and output sequences using pre-trained embeddings.

4. Model Architecture:
 - Build a sequence-to-sequence model for text summarization.
 - Create an encoder using LSTM layers (or Bidirectional LSTM layers) to process input sequences.
 - Create a decoder using LSTM layers (or Bidirectional LSTM layers) for generating output sequences.
 - Use Time Distributed layer for generating a distribution over the output vocabulary at each time step.

5. Model Compilation and Training:
 - Compile the model with appropriate loss function and optimizer.
 - Train the model using the preprocessed input and output sequences.
 - Use early stopping and learning rate reduction callbacks during training.

6. Model Evaluation and Visualization:

- Plot accuracy and loss curves during training.
- Save the trained model and training history using pickle.

7. Data Export and Conversion:
- Convert the Jupyter Notebook to a script using the `nbconvert` tool.

Please note that this is a brief overview of the code's functionality. The provided code involves various steps such as data loading, text preprocessing, model construction, training, and evaluation. It's recommended to go through each part of the code carefully to understand the implementation details and how each step contributes to building a text summarization model.

1. **Hardware Requirements :**
   (a) **Processor :** Intel Core i5 processor with minimum 2.0 GHz speed.
   (b) **RAM :** Minimum 8 GB RAM or above.
2. **Software Requirements :**
   (a) **Operating System :** Windows / Linux
   (b) Colab search engine and Jupyter notebook(Anaconda)
   (c) Python 3.6
   (d) TensorFlow
   (e) GloVe Embeddings (Pre-trained Word Vectors)
   (f) NLTK

Chapter 2

Overview

Natural Language Processing (NLP) is a field at the intersection of artificial intelligence (AI) and linguistics that focuses on enabling computers to understand, interpret, and generate human language in a way that is both meaningful and useful. The ultimate goal of NLP is to bridge the gap between human communication and computer understanding, allowing machines to interact with humans in a more natural and intuitive way.

Here's an overview of NLP:

1. Text Understanding and Interpretation:

NLP involves developing algorithms and models that allow computers to comprehend and extract meaning from text data. This includes tasks such as:

- Tokenization: Breaking text into individual words or tokens.

- Part-of-Speech Tagging: Identifying the grammatical parts of speech (noun, verb, adjective, etc.) of each word.

- Named Entity Recognition: Identifying entities like names of people, places, organizations, and dates in text.

- Sentiment Analysis: Determining the emotional tone or sentiment expressed in a piece of text (positive, negative, neutral).

2. Language Generation:

NLP also involves generating human-like text based on given prompts or contexts. This includes tasks such as:

- Text Generation: Creating coherent and contextually appropriate sentences or paragraphs.

- Machine Translation: Automatically translating text from one language to another.

- Text Summarization: Condensing large bodies of text into shorter summaries while preserving key information.

3. Speech Recognition and Synthesis:

NLP extends beyond written text to spoken language. Speech recognition involves converting spoken words into written text, while speech synthesis generates human-like speech from text. These technologies power applications like voice assistants and speech-to-text systems.

4. Question Answering:

NLP enables computers to understand questions posed in natural language and provide relevant answers. This includes tasks like:

- Information Retrieval: Extracting information from large text corpora to answer questions.

- Reading Comprehension: Understanding a given passage and answering questions about its content.

5. Language Models and Machine Learning:

Many NLP tasks are tackled using machine learning techniques, particularly deep learning. Language models like Transformers have shown remarkable performance in various NLP tasks by learning contextual relationships between words and generating coherent text.

6. Challenges in NLP:

NLP faces challenges due to the complexity and ambiguity of human language. These challenges include word sense disambiguation (determining the correct meaning of a word in context), handling figurative language, understanding context, and dealing with languages with different structures.

7. Applications:

NLP has a wide range of applications, including:

- Chatbots and Virtual Assistants: Providing automated customer support and information retrieval.

- Sentiment Analysis: Monitoring public opinion on social media or analyzing customer feedback.

- Language Translation: Enabling communication across language barriers.

- Text Summarization: Generating concise summaries from long articles.

- Healthcare: Extracting valuable insights from medical records and research papers.

Overall, NLP plays a crucial role in enabling machines to understand and communicate with humans through natural language, leading to advancements in various industries and enhancing user experiences.

Chapter 3

3.1 FUNDAMENTALS OF NATURAL LANGUAGE PROCESSING

Natural Language Processing (NLP) is a field of artificial intelligence and linguistics that focuses on enabling computers to understand, interpret, and generate human language. NLP involves a range of techniques and methods to process, analyze, and derive meaning from textual data. Here are some fundamentals of NLP:

1. Tokenization:

Tokenization is the process of breaking down a text into individual words or tokens. It's the first step in NLP and forms the basis for further analysis.

2. Part-of-Speech Tagging:

Part-of-speech tagging involves labeling each word in a text with its corresponding grammatical part of speech (e.g., noun, verb, adjective). This helps in understanding the syntactic structure of a sentence.

3. Named Entity Recognition (NER):

NER is the identification of named entities (such as names of people, places, organizations, dates) in a text. It's crucial for information extraction and understanding the context of a text.

4. Text Classification:

Text classification involves categorizing text documents into predefined classes or categories. It's used for sentiment analysis, spam detection, topic categorization, etc.

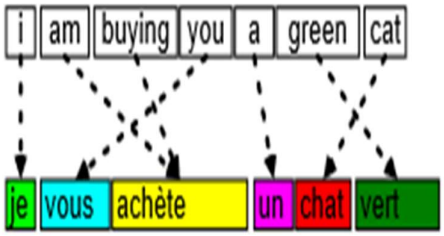
5. Language Modeling:

Language models are trained to predict the next word in a sequence of words. They're the basis for many NLP tasks, including machine translation, text generation, and more.

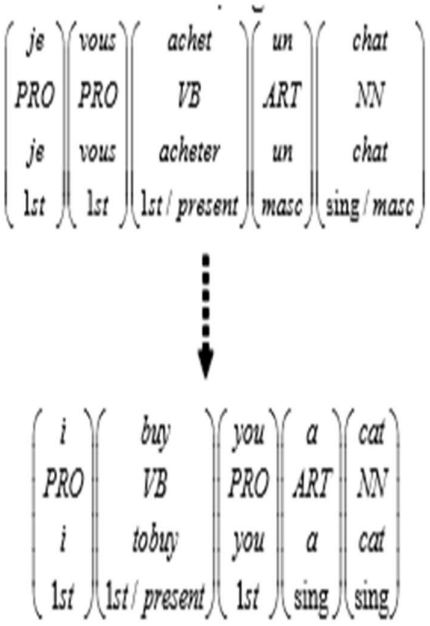
6. Machine Translation:

Machine translation involves automatically translating text from one language to another. Neural machine translation models have significantly improved translation quality.

Translate:



using phrase dictionary:



7. Text Generation:

Text generation involves producing coherent and contextually relevant text. It's used in chatbots, language generation, and creative writing applications.

8. Speech Recognition:

While not strictly textual, speech recognition converts spoken language into written text. It's used in voice assistants, transcription services, and more.

9. Dependency Parsing:

Dependency parsing is the process of analyzing the grammatical structure of a sentence to determine the relationships between words.

10. Coreference Resolution:

Coreference resolution identifies when two or more expressions in a text refer to the same entity. This is important for maintaining context and coherence.

11. Preprocessing and Cleaning:

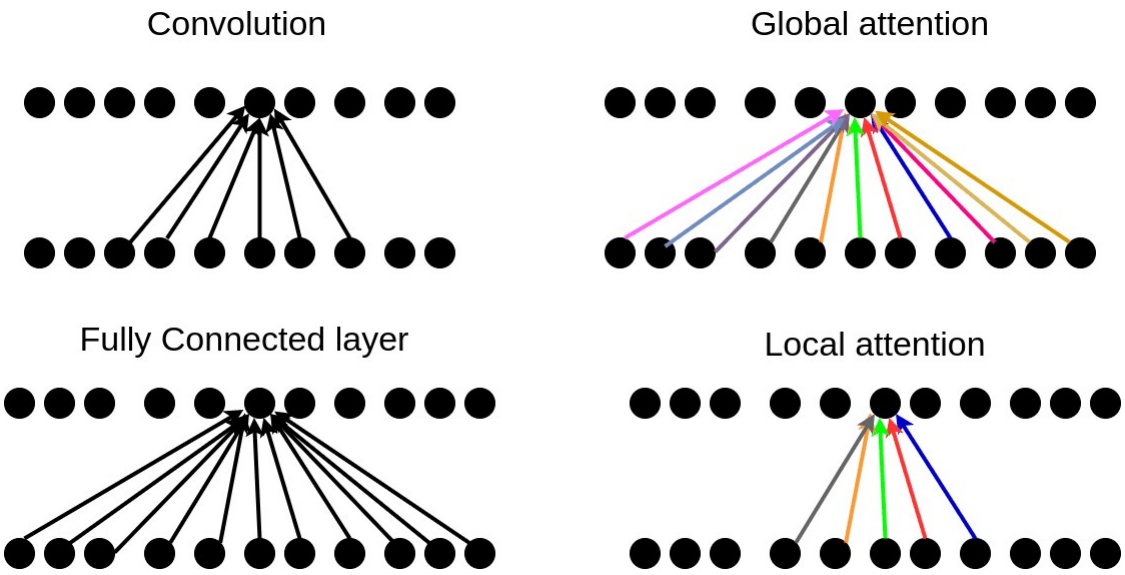
Text data often requires preprocessing, which includes tasks like removing stopwords, stemming, and handling punctuation to prepare it for analysis.

12. Word Embeddings:

Word embeddings are vector representations of words that capture semantic relationships. They're used to enhance the understanding of words by machines.

13. Attention Mechanisms:

Attention mechanisms allow models to focus on different parts of the input text while making predictions. They're crucial for tasks like machine translation and text summarization.



14. Text Summarization:

Text summarization involves condensing a longer text into a shorter summary while retaining its key information. It's used in news articles and document summarization.

These fundamentals represent the building blocks of NLP, and advances in deep learning, neural networks, and large datasets have significantly improved the capabilities of NLP models in recent years.

Chapter 4

Challenges of Abstractive Text Summarizer

Abstractive text summarization, which involves generating a concise and coherent summary by interpreting and rephrasing the source text, poses several challenges due to its complex nature. Some of the key challenges in abstractive text summarization include:

1. Content Selection:
Identifying the most relevant and important information from the source text while filtering out irrelevant or redundant details is a challenge. Models need to grasp the core meaning of the text to create a meaningful summary.

2. Coherence and Fluency:
Generating summaries that read naturally and maintain coherent flow is difficult. Ensuring that the generated summary makes sense and sounds human-like can be a challenge, especially when rephrasing sentences.

3. Paraphrasing and Rewriting:
Abstractive summarization involves rewriting sentences to create a concise summary. Achieving accurate paraphrasing while preserving the original meaning is challenging, as it requires a deep understanding of language nuances.

4**. Vocabulary and Language Variation:**
Summaries should use appropriate vocabulary and language consistent with the domain and style of the source text. Handling domain-specific terms, idiomatic expressions, and different writing styles can be complex.

5. Long-range Dependencies:
To create coherent summaries, models need to capture long-range dependencies and connections between different parts of the source text. Maintaining context and continuity is challenging, especially for longer documents.

6. Extractive vs. Abstractive Trade-off:
Deciding when to extract a sentence directly from the source text and when to generate an abstractive sentence is a challenge. Striking the right balance between these two approaches is important for creating informative and concise summaries.

7. Handling Unseen Data:
Models trained on specific domains or datasets may struggle when summarizing texts from new or unfamiliar domains. Adapting to diverse topics and generating accurate summaries for unseen data is a challenge.

8. Data Quality and Quantity:
Abstractive summarization models require large amounts of high-quality training data. Ensuring that the data covers a wide range of topics and writing styles can be a challenge.

9. Evaluation Metrics:
Evaluating the quality of abstractive summaries is challenging, as there may be multiple valid ways to rephrase and summarize a text. Common metrics like ROUGE (Recall-Oriented Understudy for Gisting Evaluation) have limitations in capturing the quality of abstractive summaries.

Addressing these challenges requires a combination of advanced natural language understanding, generation techniques, and model training strategies. Ongoing research and advancements in deep learning and NLP are aimed at overcoming these challenges and improving the quality of abstractive text summarization.

Chapter 5

Program Execution Flow

Step 1. Importing Libraries
Step 2. Installing Required Packages
Step 3. Printing Contractions.
Step 4. Loading Data
Step 5. Text Preprocessing
Step 6. Cleaned Data
Step 7. Word Cloud Generation
Step 8. Data Length Analysis
Step 9. Word Count Percentage Analysis.
Step 10. Data Trimming
Step 11. Tokenizer and Padding.
Step 12. Building the Model
Step 13. Training the Model
Step 14. Plotting Results
Step 15. Saving and Loading Trained Model.

Chapter 6

Text Summarizer

In this chapter, We discuss how humans are inundated with vast amounts of textual data and the necessity for automated techniques that can extract essential information. We explore the applications of summarization in various domains, such as news articles, research papers, social media, and legal documents. The limitations of extractive summarization, where sentences are directly selected from the input text, are elucidated. This chapter establishes the rationale for abstractive summarization techniques, which have the potential to generate more coherent and human-like summaries.

6.1 Challenges of Data Cleaning:

1. Data Collection

2. Data Inspection and Understanding

3. Data Transformation

4. Handling Missing Data

5. Data Deduplication

6. Data Validation

7. Handling Inconsistencies

8. Dealing with Text Data

9. Addressing Noisy Data

10. Documenting Changes

6.2 Literature Review

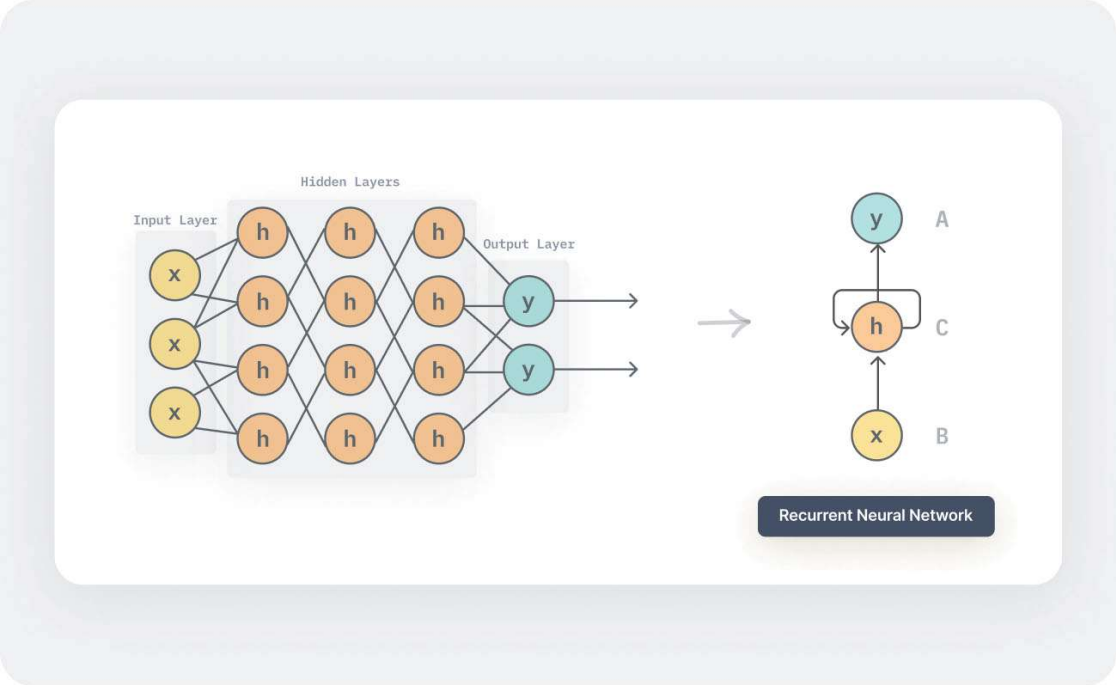
In this comprehensive literature review, we delve into the evolution of abstractive text summarization techniques using NLP. We begin by discussing early methods that relied on rule-based approaches and progress to modern machine learning techniques. We explore various neural network architectures used for sequence generation, such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, and how they form the foundation of abstractive summarization. Furthermore, we delve into the breakthrough of attention mechanisms, which revolutionized how models focus on relevant parts of the input text during summary generation. We discuss benchmark datasets used for evaluation, including the CNN/Daily Mail dataset and the Gigaword dataset, and delve into evaluation metrics like ROUGE (Recall-Oriented Understudy for Gisting Evaluation) and BLEU (Bilingual Evaluation Understudy)..

6.2 RNN

A Recurrent Neural Network (RNN) is a type of artificial neural network designed for processing sequences of data, where each element in the sequence is not treated in isolation but is influenced by the previous elements. RNNs are particularly well-suited for tasks involving sequences, such as natural language processing, speech recognition, time series analysis, and more. The architecture of an RNN is designed to handle sequential data by maintaining an internal memory state.

Basic Architecture:

At its core, an RNN consists of a set of interconnected nodes, or units, arranged in a sequential manner. Each unit takes an input and produces an output along with updating its internal state. The key idea is that the output of a unit at time step t becomes an input to the same unit at time step t+1. This looping mechanism enables the network to maintain information from previous time steps.



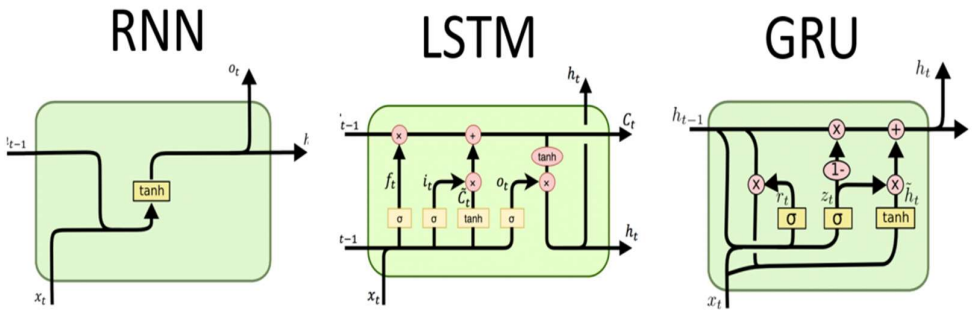
Recurrent Units:

The basic building block of an RNN is the recurrent unit, which takes three main inputs

1. Input (x_t): The input data at the current time step t.

2. Hidden State (h_t-1): The internal memory state of the unit from the previous time step t-1.

3. Previous Output (y_t-1): The output of the unit from the previous time step t-1 (not always used).



The recurrent unit computes the following:

1. Hidden State Update: It combines the current input and the previous hidden state to compute a new hidden state for the current time step: h_t = f(W_hh * h_t-1 + W_xh * x_t + b_h).

2. Output: The hidden state can be used to compute the output at the current time step: y_t = f(W_hy * h_t + b_y), where f is typically an activation function like the sigmoid or tanh function.

3.Training: During training, the parameters (weights and biases) of the RNN are updated using gradient descent and backpropagation through time (BPTT). BPTT extends the regular backpropagation algorithm to handle sequences by considering the effect of the loss function across all time steps.

4.Challenges: However, basic RNNs suffer from some limitations, particularly in capturing long-range dependencies. As the network processes longer sequences, the gradient can either explode or vanish, which leads to difficulties in learning long-term patterns. To address this, several advanced RNN architectures have been developed, including Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs). These architectures incorporate gating mechanisms that control the flow of information, making them better at handling long-range dependencies.

5.LSTM and GRU:

LSTM and GRU are popular variants of RNNs designed to mitigate the vanishing gradient problem and improve the learning of long-range dependencies. They achieve this through mechanisms that regulate the flow of information, allowing the network to remember or forget information over long sequences.

- LSTM (Long Short-Term Memory): LSTM units have a more complex structure with three interacting gates: forget gate, input gate, and output gate. These gates control what information to forget, what new information to add, and what information to output, respectively.

- GRU (Gated Recurrent Unit): GRU units simplify the LSTM architecture by combining the hidden state and memory cell into a single hidden state. They use two gates: reset gate and update gate. The reset gate determines how to combine the new input with the previous hidden state, while the update gate controls how much of the previous state to keep.

Both LSTM and GRU networks have proven effective in capturing long-range dependencies and are widely used in various sequence-based tasks.

In summary, RNNs are neural networks specifically designed to handle sequential data by maintaining an internal memory state that evolves as the network processes the sequence. While basic RNNs have limitations in capturing long-term dependencies, LSTM and GRU architectures have emerged as powerful solutions to address these issues and excel in tasks involving sequences.

6.3 LSTM

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to capture and model sequential data by addressing the vanishing gradient problem that traditional RNNs often face. LSTMs are particularly effective in processing and learning from sequences of data, making them popular in various tasks such as natural language processing, speech recognition, time series analysis, and more. Here's a comprehensive explanation of LSTM architecture:

1. LSTM Introduction:

LSTMs were introduced to address the vanishing gradient problem and to enable RNNs to better capture long-term dependencies. They achieve this through three key components: the input gate, the forget gate, and the output gate. These gates regulate the flow of information into and out of the LSTM cell.

2. LSTM Architecture:

The LSTM architecture consists of the following components:

a. Cell State : The cell state serves as the memory of the LSTM cell, allowing it to maintain information over long sequences. The cell state can be modified through various gates.

b. Hidden State : The hidden state is the output of the LSTM cell at each time step and contains information relevant to the current prediction.

c. Input Gate : The input gate controls how much new information is added to the cell state. It takes the current input and the previous hidden state as inputs and produces a value between 0 and 1 for each element in the cell state.

d. Forget Gate : The forget gate determines how much information from the previous cell state should be retained. It takes the current input and the previous hidden state as inputs and produces a value between 0 and 1 for each element in the cell state.

e. Output Gate : The output gate controls how much of the cell state is exposed to the hidden state. It takes the current input and the previous hidden state as inputs and produces a value between 0 and 1 for each element in the cell state.

f. Candidate State ($\hat{C}_t$): The candidate state represents the new information that could be added to the cell state. It is computed using the current input and the previous hidden state and is then modulated by the input gate.

3. LSTM Operation:

Here's how the LSTM cell operates at each time step:

- Calculate the input gate (i_t), forget gate (f_t), and output gate (o_t) values using the current input and the previous hidden state.

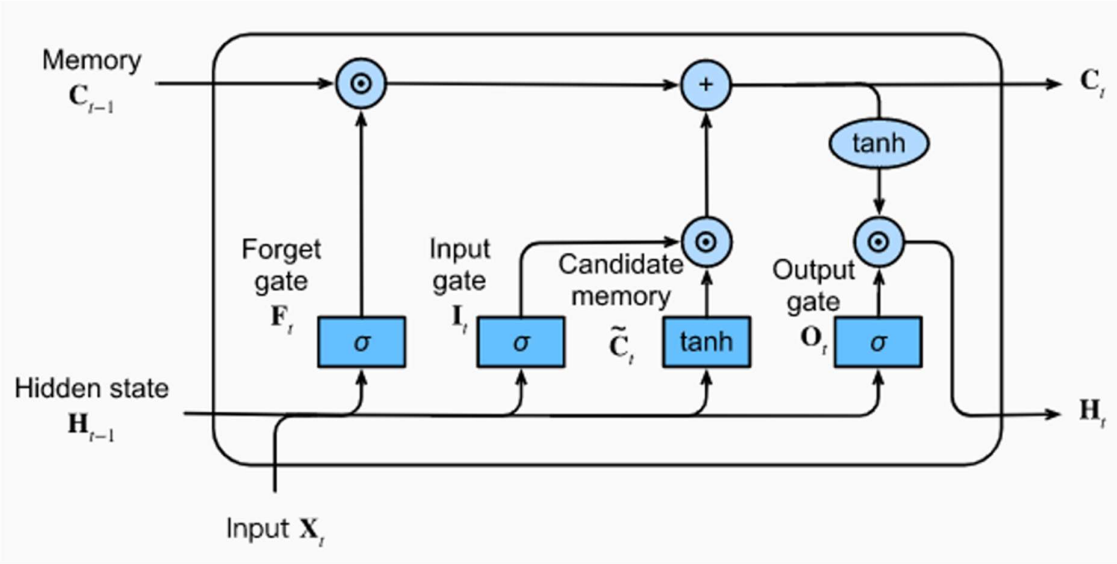
- Calculate the candidate state ($\hat{C}_t$) using the current input and the previous hidden state.

- Update the cell state (C_t) by combining the forget gate (f_t) with the previous cell state and adding the input gate-modulated candidate state.

- Calculate the new hidden state (h_t) by applying the output gate (o_t) to the updated cell state.

4. Training and Backpropagation:

LSTMs are trained using backpropagation through time (BPTT), a variant of gradient descent. The gradients are calculated over the entire sequence, allowing the network to learn from past inputs and modify its parameters accordingly.

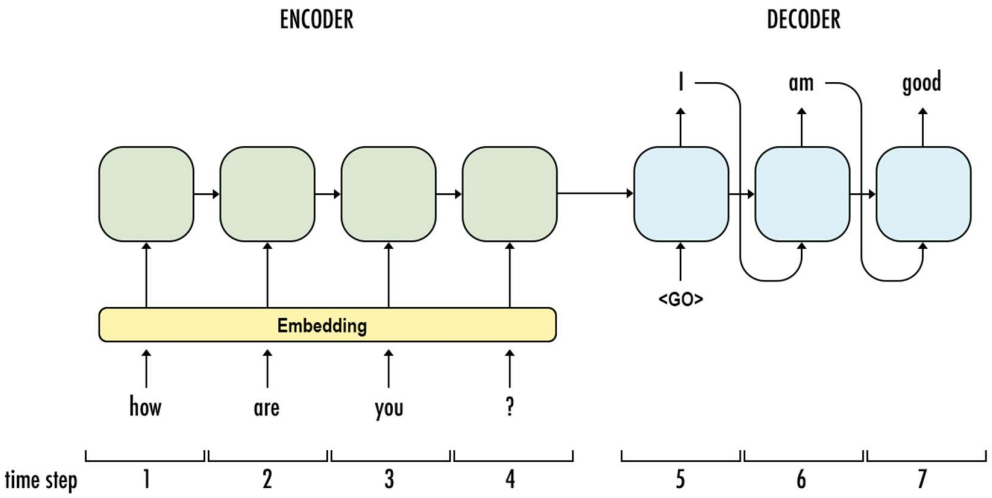


In summary, the LSTM architecture's ability to control the flow of information through input, forget, and output gates enables it to capture long-term dependencies in sequential data, making it a powerful tool for tasks involving sequences.

6.4 Sequence-to-Sequence Model

Sequence-to-Sequence (Seq2Seq) models are a type of neural network architecture designed for tasks that involve transforming an input sequence into an output sequence. These models are widely used in various natural language processing (NLP) tasks, such as machine translation, text summarization, speech recognition, and more. Here's a brief explanation of Sequence-to-Sequence models:

1. Basic Structure:
Seq2Seq models consist of two main components: an encoder and a decoder. The encoder processes the input sequence and compresses its information into a fixed-size vector called the "context" or "thought" vector. The decoder takes this context vector as input and generates the output sequence step by step.



2. Encoder:

The encoder's role is to process the input sequence element by element and capture the relevant information in a compact representation. It typically utilizes recurrent neural networks (RNNs) like Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) to read the input sequence and update its hidden states at each time step. The final hidden state or the collection of hidden states is then used as the context vector that encodes the input sequence.

3. Context Vector:
The context vector serves as a summary of the input sequence and contains the crucial information that the decoder needs to generate the output sequence. The decoder uses this vector to initialize its hidden state and generate the output sequence step by step.

4. Decoder:
The decoder is responsible for generating the output sequence based on the context vector and the previously generated tokens. It also utilizes an RNN, usually of the same type as the encoder's RNN, to generate each token of the output sequence one by one. At each time step, the decoder's hidden state is updated using the context vector and the previously generated token.

5. Training:
During training, Seq2Seq models are optimized to minimize the difference between the predicted output sequence and the actual target sequence. This involves using techniques like teacher forcing, where the correct previous token is fed as input to the decoder at each step, instead of using the predicted token.

6. Variations:
Several variations of Seq2Seq models have been introduced to address their limitations, such as attention mechanisms, which allow the decoder to focus on different parts of the

input sequence while generating each token. Transformer-based models like the Transformer and its variants have also gained popularity due to their parallelizability and improved performance in sequence tasks.

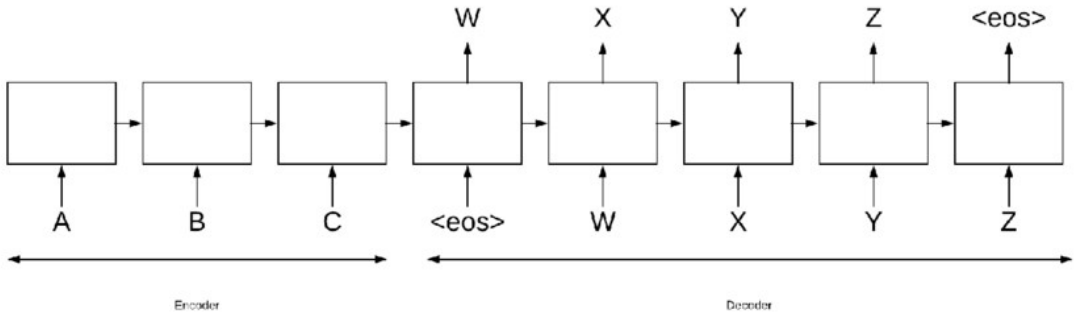
In summary, Sequence-to-Sequence models are neural network architectures designed to handle tasks that involve converting one sequence into another. They consist of an encoder to process the input sequence and a decoder to generate the output sequence, making them highly versatile for various sequence-based applications.

Chapter 7

Models

7.1 Sequence-to-Sequence (Seq2Seq) Model:

A Sequence-to-Sequence model is a type of neural network architecture designed to handle input and output sequences of varying lengths. It's commonly used for tasks like machine translation, text summarization, and speech recognition, where the input and output can have different lengths.



Architecture of Seq2Seq Model with Just LSTMs:

1. Encoder:

- The input sequence (e.g., a sentence in machine translation) is tokenized and represented as a sequence of word embeddings.

- These embeddings are fed into the LSTM layers in the encoder. The LSTM layers capture the sequential information and hidden state of the input sequence.

- The final hidden state of the encoder LSTM is taken as the context vector. This context vector encapsulates the information from the entire input sequence in a fixed-size representation.

2. Decoder:

- The context vector from the encoder is used as the initial hidden state of the decoder LSTM. This allows the decoder to start generating the output sequence with the relevant context.

- The decoder generates the output sequence one token at a time. At each time step, it predicts the next token based on the previous token and the current hidden state.

- The predicted token is then appended to the output sequence, and the process is repeated iteratively until an end-of-sequence token is generated or a maximum sequence length is reached.

3. Training:

- During training, the model is provided with paired input-output sequences (source and target sequences).

- The encoder processes the source sequence to create the context vector.

- The decoder takes the context vector and generates the target sequence step by step, conditioned on the previous tokens.

4. Loss Function:

- The model's predicted sequence is compared to the actual target sequence using a loss function such as cross-entropy loss.

- The gradients are backpropagated through the model, and the model's parameters are updated using optimization algorithms like stochastic gradient descent (SGD) or Adam.

Advantages of Seq2Seq Model with Just LSTMs:

1.Variable-Length Input and Output: Seq2Seq models can handle input and output sequences of different lengths, making them suitable for tasks where the length of the input and output varies, such as translation.

2.Sequential Information: LSTMs are capable of capturing sequential dependencies in the data, which is crucial for tasks involving language, where the order of words matters.

3.Contextual Understanding: The context vector generated by the encoder LSTM carries important information about the entire input sequence, helping the decoder generate meaningful output.

Challenges and Considerations:

1.Vanishing Gradient: LSTMs can still suffer from vanishing gradient problems, especially in longer sequences. Techniques like gradient clipping or using more advanced LSTM variants (e.g., GRUs) can help mitigate this.

2.Limited Context: LSTMs have a fixed-size context vector, which can limit the model's ability to capture very long-range dependencies in the input sequence.

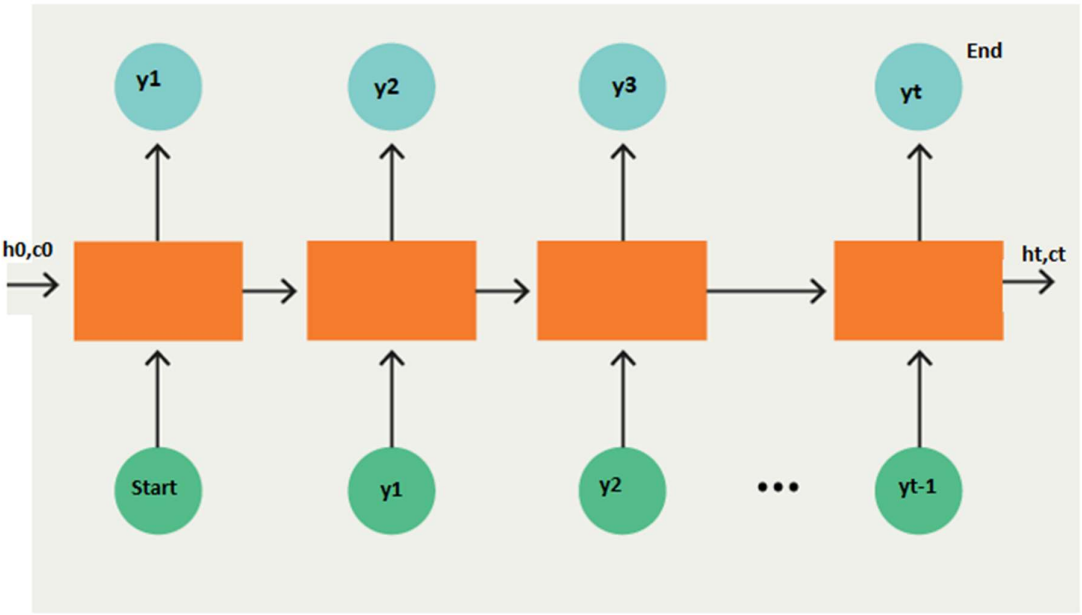
3.Attention Mechanisms: To address the limited context issue, attention mechanisms are often used in conjunction with Seq2Seq models. Attention mechanisms allow the decoder to focus on different parts of the input sequence at different time steps.

In summary, the Seq2Seq Model with Just LSTMs is a powerful architecture for tasks involving sequential data, but it might struggle with very long sequences or tasks where more advanced techniques like attention mechanisms are required to handle complex dependencies.the measurement is trusted less and the predicted value is trusted more and more.

Certainly, let's dive deeper into the architecture and functioning of a Seq2Seq Model with Bidirectional LSTMs.

7.2 Sequence-to-Sequence (Seq2Seq) Model:

A Sequence-to-Sequence model is a type of neural network architecture designed to handle input and output sequences of varying lengths. It's commonly used for tasks like machine translation, text summarization, and speech recognition, where the input and output can have different lengths.



Architecture of Seq2Seq Model with Bidirectional LSTMs:

1. Encoder:

- The input sequence (e.g., a sentence in machine translation) is tokenized and represented as a sequence of word embeddings.

- Unlike the unidirectional LSTM used in the previous model, here we use bidirectional LSTMs in the encoder. A bidirectional LSTM consists of two LSTM layers: one processes the sequence from left to right, and the other processes it from right to left.

- The bidirectional LSTMs capture both forward and backward contextual information of each token in the input sequence.

- The outputs of both the forward and backward LSTMs at each time step are concatenated to create a richer context representation for that time step.

2. Context Vector:

- The final hidden states of both the forward and backward LSTMs are concatenated to form the context vector for that time step. This context vector captures the comprehensive information about the token at that time step, considering both past and future contexts.

3. Decoder:

- The context vector from the bidirectional encoder is used as the initial hidden state of the decoder LSTM. This context provides a strong foundation for generating the output sequence.

- Similar to the previous model, the decoder generates the output sequence step by step, predicting the next token based on the previous token and the current hidden state.

4. Training:

- During training, the model is provided with paired input-output sequences (source and target sequences).

- The encoder processes the source sequence with bidirectional LSTMs to create the context vectors.

- The decoder takes the bidirectional context vectors and generates the target sequence iteratively.

5. Loss Function:

- The model's predicted sequence is compared to the actual target sequence using a loss function such as cross-entropy loss.

- Backpropagation is performed to update the model's parameters.

Advantages of Seq2Seq Model with Bidirectional LSTMs:

- Enhanced Context: By using bidirectional LSTMs in the encoder, the model captures both past and future contextual information for each token. This helps the model generate more accurate and contextually relevant outputs.

- Better Understanding of Context: Bidirectional LSTMs are particularly effective for tasks where the meaning of a token is influenced by the tokens that come both before and after it in the sequence. For example, in language translation, the translation of a word may depend on the surrounding words in both directions.

Challenges and Considerations:

Complexity: Bidirectional LSTMs introduce more complexity compared to unidirectional LSTMs, which can lead to longer training times and a higher risk of overfitting if the dataset is small.

- Increased Computation: Since bidirectional LSTMs process the sequence from both directions, they require more computational resources compared to unidirectional LSTMs.

- Attention Mechanisms: While bidirectional LSTMs capture a broader context, attention mechanisms can still be valuable to focus on specific parts of the input sequence that are most relevant for generating the current output.

In summary, the Seq2Seq Model with Bidirectional LSTMs is a more advanced architecture that takes advantage of bidirectional information flow to capture richer context and improve the model's ability to handle tasks involving complex dependencies in sequences.
Certainly, let's delve into the details of a Hybrid Seq2Seq Model.

7.3 Hybrid Seq2Seq Model:

The Hybrid Seq2Seq Model combines the strengths of both unidirectional and bidirectional LSTMs in the encoder and decoder parts of the Sequence-to-Sequence architecture. This approach aims to capture a comprehensive context for input sequences while maintaining computational efficiency.

Architecture of Hybrid Seq2Seq Model:

1. Encoder:

- The input sequence is tokenized and represented as a sequence of word embeddings.

- Unlike the previous models, the hybrid architecture employs a Bidirectional LSTM (BiLSTM) in the encoder.

- The BiLSTM processes the input sequence in both forward and backward directions, creating two sets of hidden states: forward and backward.

2. Context Vector Generation:

- For each time step, the final hidden states from both the forward and backward passes of the BiLSTM are concatenated to create the context vector for that step.

- This context vector contains information from both past and future contexts of the input sequence, making it rich and informative.

3. Decoder:

- Similar to the previous models, the decoder consists of a unidirectional LSTM.

- The context vector obtained from the BiLSTM in the encoder is used as the initial hidden state of the decoder LSTM.

- The decoder generates the output sequence step by step, using the previous token and hidden state as input.

4. Training: During training, the model is provided with paired input-output sequences (source and target sequences).

- The encoder processes the source sequence with the BiLSTM to create the context vectors.

- The decoder takes the hybrid context vectors and generates the target sequence iteratively.

5. Loss Function: The predicted sequence is compared to the actual target sequence using a loss function such as cross-entropy loss.

- Backpropagation is performed to update the model's parameters.

Advantages of Hybrid Seq2Seq Model:

- Rich Context: By combining both unidirectional and bidirectional LSTMs in the encoder, the hybrid model captures a comprehensive context for each token in the input sequence, considering both past and future contexts.

- Balanced Complexity: The hybrid approach strikes a balance between capturing richer context and computational efficiency. It is often less computationally intensive than a full bidirectional architecture.

- **Effective for Complex Dependencies:** The hybrid model is effective for tasks where certain tokens depend on both previous and future tokens in the sequence to be correctly understood or generated.

Challenges and Considerations:

- Complex Architecture: The hybrid model has a more complex architecture compared to a simple unidirectional model, which can increase training time and model complexity.

- Tuning Parameters: The hybrid model introduces additional hyperparameters that need to be tuned, such as the number of hidden units in the encoder and decoder.

- Data Requirements: A hybrid model might require a larger dataset to effectively learn the parameters and exploit the additional complexity.

In summary, the Hybrid Seq2Seq Model combines the benefits of unidirectional and bidirectional LSTMs to capture a holistic context while maintaining computational efficiency. This architecture is particularly useful for tasks that demand a nuanced understanding of input sequences with complex dependencies.

Chapter 8

Proposed Methodology and Simulation

8.1 Designed Overview

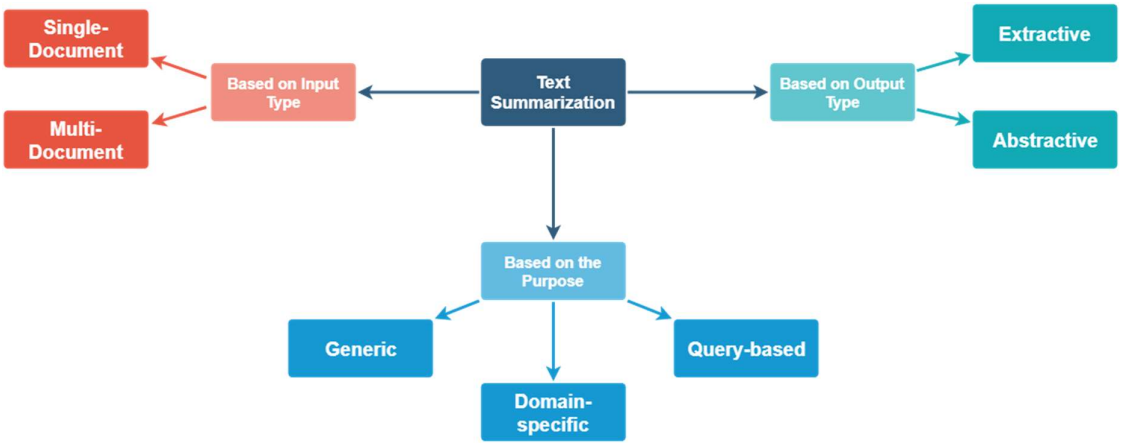


Figure 8.1: Flowchart of Text Summarization using NLP

1. **Data Cleaning**

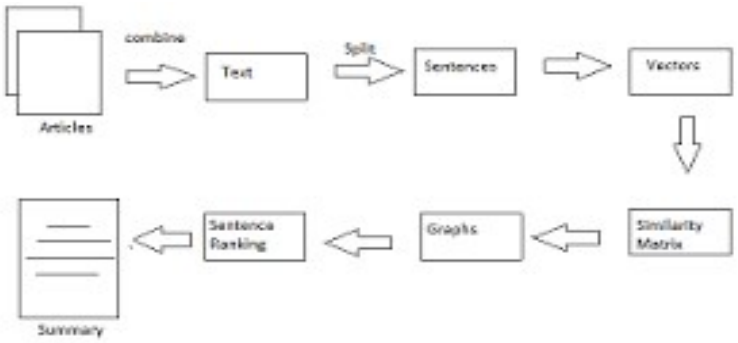


Figure 8.2: Data Cleaning

2. **Classification**

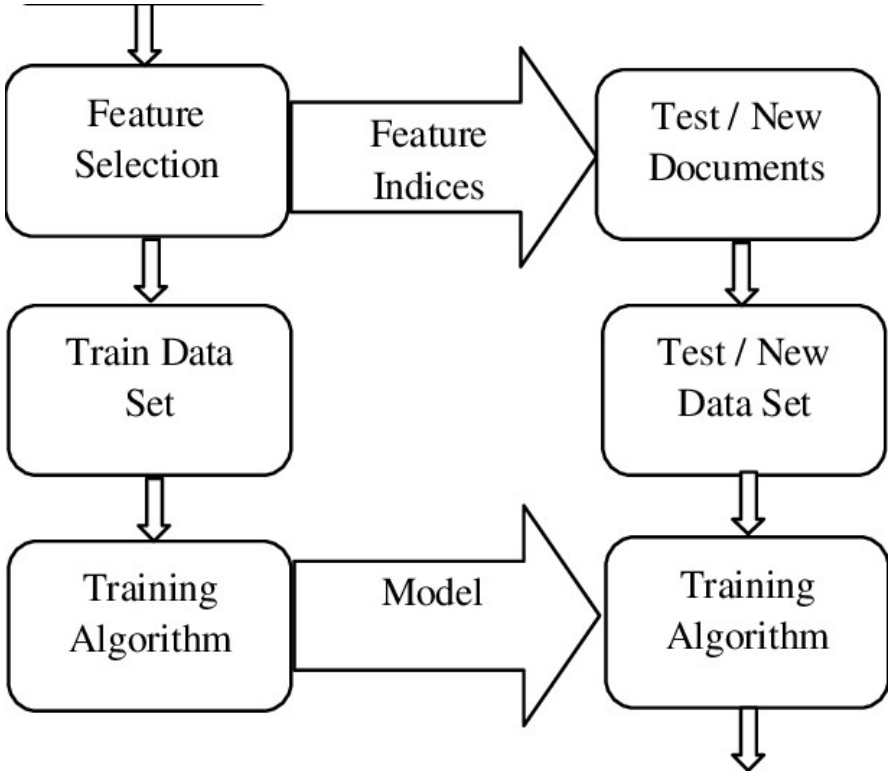
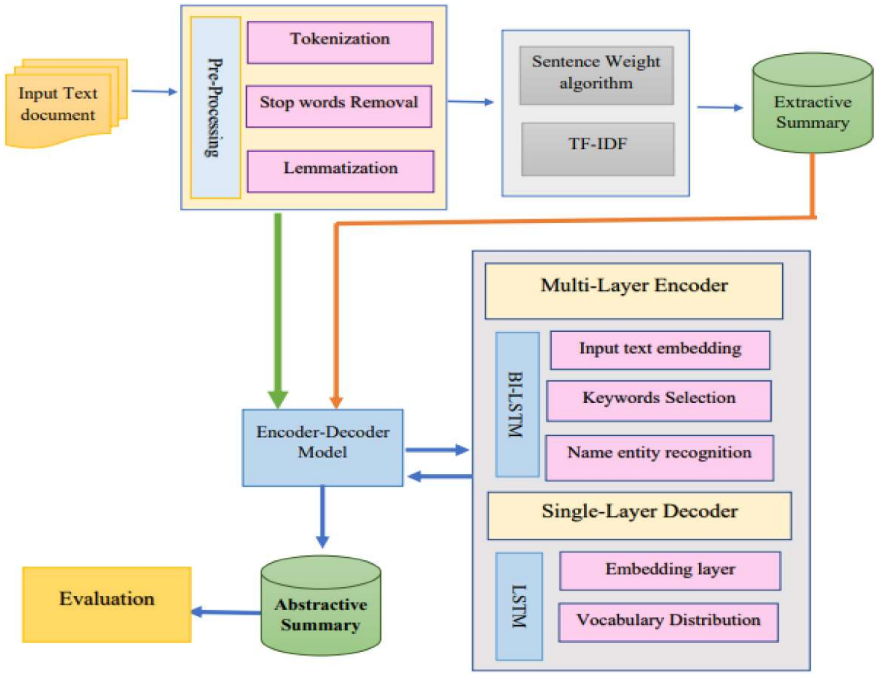


Figure 8.3: summary of text

3. **Model Flowchart**



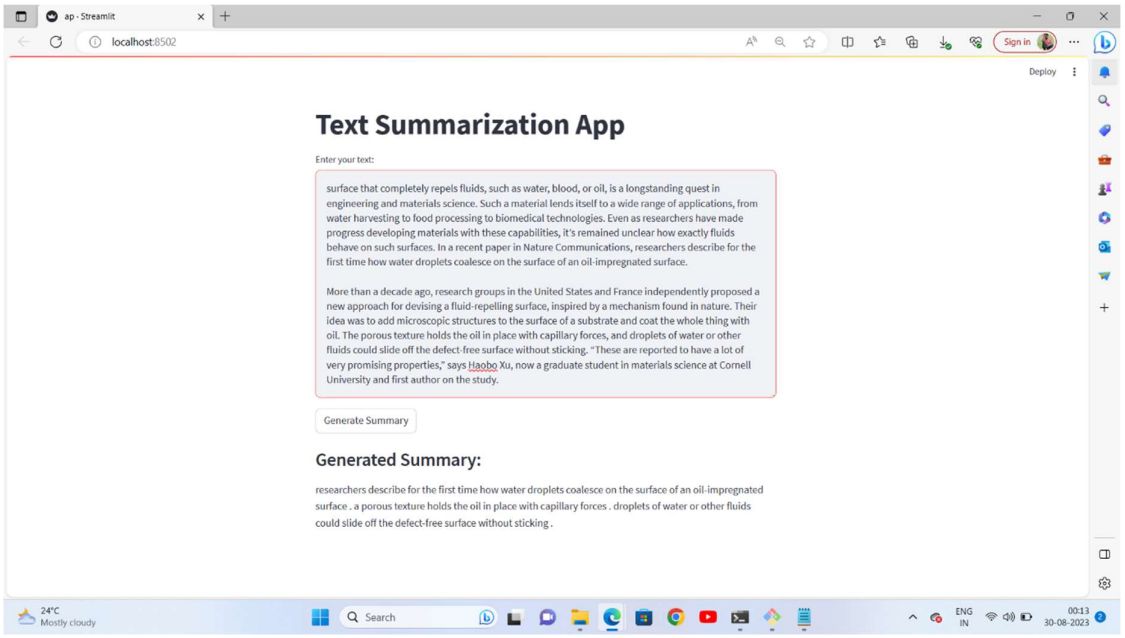
8.2 Design Requirements

1. **Input Text (Dataset Used) :** https://cnnDaily.mail/data/MOT17Det/

2. **Predicting Summary :** Transformers (Encoder & Decoder)

3. **Associating text summaries with Different Frames :** Seq2Seq Model Using LSTM Algorithm

OUTPUT


ap · Streamlit
localhost:8502
Sign in
Deploy
Text Summarization App
Enter your text:
surface that completely repels fluids, such as water, blood, or oil, is a longstanding quest in engineering and materials science. Such a material lends itself to a wide range of applications, from water harvesting to food processing to biomedical technologies. Even as researchers have made progress developing materials with these capabilities, it's remained unclear how exactly fluids behave on such surfaces. In a recent paper in Nature Communications, researchers describe for the first time how water droplets coalesce on the surface of an oil-impregnated surface.
More than a decade ago, research groups in the United States and France independently proposed a new approach for devising a fluid-repelling surface, inspired by a mechanism found in nature. Their idea was to add microscopic structures to the surface of a substrate and coat the whole thing with oil. The porous texture holds the oil in place with capillary forces, and droplets of water or other fluids could slide off the defect-free surface without sticking. "These are reported to have a lot of very promising properties," says Haobo Xu, now a graduate student in materials science at Cornell University and first author on the study.
Generate Summary
Generated Summary:
researchers describe for the first time how water droplets coalesce on the surface of an oil-impregnated surface . a porous texture holds the oil in place with capillary forces . droplets of water or other fluids could slide off the defect-free surface without sticking .
24°C
Mostly cloudy
Search
ENG
IN
00:13
30-08-2023

Chapter 9

CONCLUSION

We have done implementation of a text summarization system using a Sequence-to-Sequence (seq2seq) model based on Long Short-Term Memory (LSTM) networks. The system takes lengthy input texts and generates concise summaries, facilitating the understanding of complex content. By incorporating data preprocessing, tokenization, embedding, and a well-designed architecture, the code demonstrates effective utilization of TensorFlow's TPU capabilities for efficient training. The integration of monitoring techniques, regularization strategies, and modular code structure further enhances the model's performance and maintainability. With the ability to visualize training progress and generate word clouds, this code provides a comprehensive solution for text summarization tasks while offering room for hyperparameter tuning and experimentation to optimize model outcomes..

Bibliography

[1] Rush, A. M., Chopra, S., & Weston, J. (2015). A neural attention model for abstractive sentence summarization. In Proceedings of the 2015 conference on empirical methods in natural language processing (EMNLP) (pp. 379-389).

[2] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

[3] Nallapati, R., Zhai, F., & Zhou, B. (2017). Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (Vol. 1, pp. 781-791).