Text Summarization

Konuralp Dalkılınç
Işık University
Istanbul, Turkey
konuralp.dalkilinc@gmail.com

***Abstract*—The main goal of the text summarization model is to create shorter and meaningful summaries that capture the important details of the longer input. This paper tries to build a simple sequence-to-sequence model for summarization. The tokenizer and encoder-decoder architecture that utilize LSTM layers is implemented from scratch. The main dataset used for training and evaluation is Multi-XScience imported from HuggingFace. Since training is resource-expensive, Google Colab is used for training environment. The paper's goal is to establish basic summarization using simple pipelines and fundamental architectures.**

I. Introduction

With the recent developments in artificial intelligence, especially in deep learning and machine learning, many applications have become part of our daily lives. One useful example is text summarization. As the amount of information grows rapidly, a tool that can reduce the size of the content while keeping the important parts is very useful.

Sequence-to-sequence models have become the standard for abstractive text summarization, especially with the growing popularity of neural network-based models. They can create summaries at a human level or even better by capturing the underlying meaning.

This paper explores implementation of a basic sequence-to-sequence text summarization model built from scratch. Main focus is on learning the core ideas of NLP keeping it simple and clear, using pre-trained complex models as little as possible.

Tokenization is used so words can be represented, an LSTM-based encoder-decoder model is built, and word embeddings are used to give meaning to words. The goal is to understand how each part works instead of just relying on ready-made solutions.

A. Multi-XScience Dataset

The Multi-XScience dataset includes a large number of documents. These can be used to train summarization models. Multi-XScience is capable of multi-document summarization, a model that can combine information with multiple documents. The dataset has around 40,000 examples. It's split into 30,019 for training, 3,814 for validation, and 3,769 for testing. The dataset structure consists of source abstracts, source titles, the target summary, and the target paper title. The target is a summary corresponding to those documents. More detailed explanation and visualization is done in the coding part.

II. SUMMARIZATION

The formal definition of summarization in language is ”the act of expressing the most important facts or ideas about something or someone in a short and clear format”. When considering natural language processing, summarization is one of the fundamental tasks. Enabling models to give much shorter and dense output from given input.

Summaries are separated into two categories: abstractive and extractive. Abstractive summaries are able to create more human-like summaries by generating new sentences and rephrasing content, but it requires more effort on training and tuning to get factually right information. Extractive summaries, on the other hand, are rather simpler; they directly copy and paste the information in the given document.

III. TOKENIZER

A tokenizer turns text into small units, called tokens. Tokens can represent multiple entities, such as words, sub-words, and characters. Tokenization is the first step in processing text for natural language models, converting text into a format that machines can understand.

Type	**Input**	**Tokens**
Whitespace Tokenization	”I love pizza”	[”I”, ”love”, ”pizza”]
Word Tokenization	”I'm sad”	[”I”, ”m”, ”sad”]
Subword Tokenization	”unhappiness”	[”un”, ”happi”, ”ness”]
Character Tokenization	”Pizza”	[”P”, ”i”, ”z”, ”z”, ”a”]

TABLE I
DIFFERENT TYPES OF TOKENIZATION

The tokenizer in this paper processes, both the abstract and ref-abstract fields by collecting all text into one big list, converting all words to lowercase, and splitting them. Unique words are stored in a set and each is assigned a unique ID starting from 2.

'pad' and 'unk' tokens are also added to handle padding and unknown words. During the encoding process, tokenizer converts input text to token IDs. Tokenizer applies padding for shorter ones and cutting longer ones according to maximum length. Decoding process is the reverse of the encoding. Idx2Word is used for turning IDs to words.

IV. SEQ2SEQ

Sequence-to-sequence (seq2seq) is a type of neural network architecture. They are able to transform one sequence to another, meaning it can understand the context of the text and act accordingly. Using an encoder and decoder pair it can map input to output. After the encoder processes the input and compresses it to the context vector, the decoder generates an output sequence, one token at each time step. Embedding layers are used to obtain dense vector representation. These vectors are passed through the LSTM layer that capture relationships between words. This state is called the hidden state. At the end the decoder receives this hidden state and processes it at each time step.

V. Encoders

Since raw text is not usable by neural networks. Encoders convert tokens into vector representations that capture their semantic meanings and properties. Each token from the input vocabulary is mapped to vectors. These embeddings represent learned characteristics of words, allowing the model to capture semantic similarities and contextual relationship between them. As a result, closely related words came close to each other. Short-Term Memory (LSTM) network is used to act as memory. Following the embedding layer LSTM processes the vectorized tokens. LSTM captures long range relationships between sequences of data. A hidden state where each vector representation is hidden inside is created as each time step LSTM works. As a result of the combination of word embeddings and the LSTM's memory, a context-aware representation of the input is constructed.

VI. Decoders

Decoder is the second component in the encoder-decoder pair. While the encoding process includes compression of input text to vector, decoder does the opposite. It processes this vector and generates an output sequence, converting numerical token prediction back to human-readable text.

LSTM (Long Short-Term Memory) networks use the initial hidden state from the encoder. At each time step, the decoder takes a token as input and predicts the next token in the sequence until it reaches the end.. The ¡sos¿ (start of sequence) token is used to show the beginning, and the ¡eos¿ (end of sequence) token is used to represent the end or when the maximum length is reached.

A. Semantic Meanings - Vector Embeddings

Semantic meaning in natural language processing refers to the understanding of text at both the word and sentence level. Rather than focusing only on surface-level meanings, it captures deeper relationships and context, providing a more detailed and structured analysis of language. Hence, it is essential for enabling machines to understand text in a way that is close or even better than human level of understanding, including the ability to detect sarcasm or underlying meanings.

To enable machines to understand human language, vector embeddings are used. These embeddings represent words as dense numerical vectors that capture semantic features such as context, relationships, and meaning. As a result, each word is placed in a vector space where semantically related words such as 'king' and 'queen' are located close to each other, reflecting their fundamental similarity in language use.

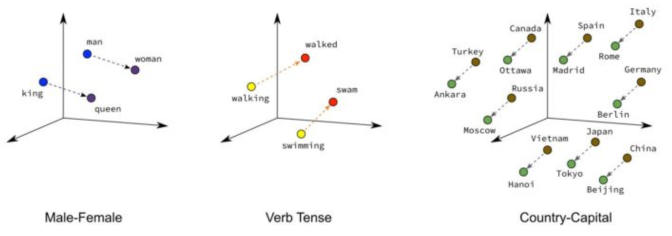


Fig. 1. representation of word embedding in vector space

VII. Parameters and Hyperparameters

A. Vocablary Size

Vocabulary refers to a set of unique tokens. Those tokens are used for understanding and generating texts. Within the tokenizer, vocabulary is created from input data. Around 250000 unique tokens are extracted. However, due to computation restrictions not all of the tokens are used.

B. Hidden Dimension

The hidden dimension refers to the size of the hidden state in the LSTM, which determines the model's capacity to store and process information. Initially, not being able to connect a GPU, ram related problems occurred forcing hidden dimensions as low as 32. At that level, the model was unable to learn anything meaningful. After resolving the connection issue, hidden dimension increased drastically positively affecting model ability to learn.

C. Embedded Dimension

The embedding dimension represents the number of values used to encode each token. It determines the level of detail in the token's vector representation. For example, a token such as 12 might be converted into a vector like [0.01, -0.32, ..., 0.76]. These details directly affect the model's ability to capture semantic meaning and relationships between words.

D. Number of Layers

Number of layers represents how many LSTM layers are stacked on top of each other.

E. Optimizer

Optimizers update model's weights and biases in order to minimize the loss function. Using backpropagation gradients are computed. Optimizers use these gradients to minimize loss. This process is crucial for models to actually learn from data. Adam Optimizer is used for this project. Fast convergence and minimal hyperparameter tuning is the reason for choosing this model.

F. Loss Function

A loss function measures how wrong your model's predictions are. It uses the difference between the model's prediction and actual outputs. This value is used for updating the model's parameters to decrease loss. This project approaches text summarization as sequence-to-sequence classification, hence Cross-Entropy Loss is chosen. Since it is widely used for multi-class classification tasks.

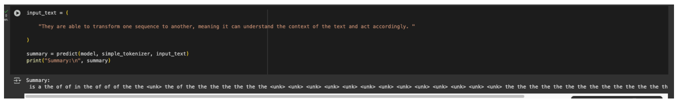
G. Batch Size

Batch size represents how many training examples are processed in one pass, affecting training speed.

H. Learning Rate

The learning rate controls how much the model updates its weights after each step. If the learning rate is too high, model overshoots. In the case of a low learning rate model learns slow or stuck at local min instead of global min.

VIII. TROUBLESHOOTING AND TUNING



```
input_text = (

    "They are able to transform one sequence to another, meaning it can understand the context of the text and act accordingly. "

)

summary = predict(model, simple_tokenizer, input_text)
print("Summary:\n", summary)
```

```
Summary:
 is a the of of in the of of of the the <unk> the of the the the the the the the <unk> <unk> <unk> <unk> <unk> <unk> <unk> <unk> <unk> <unk> <unk> <unk> <unk> the the the the the the the the the the the the the th
```

Fig. 2. Enter Caption

The first road block was not to preprocess input text. This resulted in an excessive vocabulary size of 134702. which is both impractical and ineffective for most natural language processing tasks. After completing model training, the model did not produce meaningful results, frequently repeating tokens such as 'of', 'the', and 'to'." A vocabulary limit is added considering most used words to decrease vocabulary size. Second and perhaps the most crucial mistake was to assume the model would perform well on the first attempt. As result, hyperparameters such as the learning rate, hidden dimension, and embedding dimension were adjusted and the model is trained multiple times without a systematic tuning. So hyper parameters, learning rate, hidden dimension, embedding dimension changed and trained a couple of times. Which created multiple models not capable of creating successful results.

After unsuccessful results, a new method "early stop" is added mainly to prevent overfitting during training. Thus a bottleneck architecture is created by setting the embedding dimension to 256 and the hidden dimension to 128. Also to address the model's "black-box" nature, a graph-based approach is applied to better understand the model's dynamics. However despite all efforts models output remained unsuccessful. Still failed to generate meaningful results. Indicating further architectural improvements and data handling strategies are necessary.

IX. CONCLUSION

Project's main aim is to build a text summarization model using sequence-to-sequence LSTM-based encoder-decoder architecture. From tokenizer to sequence-to-sequence class most of the architectures are implemented from scratch. While the model did not successfully generate summaries from the input documents due to technical and computational limitations, paid version of Google Colab is used to overcome issues related to RAM, CPU, and GPU. Despite these efforts, a successful model has not been created. However the process itself provided significant educational value. Fundamental concepts such as tokenization, embeddings, semantic meaning, and sequence modeling were implemented and understood throughout the project. Overall, the project laid a solid foundation for future work in natural language processing and deep learning.

REFERENCES

https://github.com/yaolu/Multi-XScience
https://developers.google.com/machine-learning/crash-course/embeddings/translating-to-a-lower-dimensional-space