**A blue shield with a cross and a book

Description automatically generated**

**Project 2 For Diploma in AI and its application in Business:**

**NLP Text Classification and Generation**

**Project Report**

*Text Preprocessing utilization and example Usage of LSTM Layer and Seq2Seq Modeling*

Prepared by: Chen Yijie

Supervisor: Zhou Changxin

Table of Contents

[Project Overview: 2](#_Toc163841414)

[Covering Concepts: 3](#_Toc163841415)

[Potential Use Case: 5](#_Toc163841416)

[Environment Used: 6](#_Toc163841417)

[Text Sentiment Classification: 6](#_Toc163841418)

[**Dataset Used:** 6](#_Toc163841419)

[**Preprocessing:** 7](#_Toc163841420)

[**Model Selection:** 8](#_Toc163841421)

[**Training Parameter:** 11](#_Toc163841422)

[**Results:** 14](#_Toc163841423)

[**Potential Improvements to be Done:** 14](#_Toc163841424)

[Text Generation 15](#_Toc163841425)

[**Dataset Used** 15](#_Toc163841426)

[**Preprocessing:** 15](#_Toc163841427)

[**Model Selection:** 17](#_Toc163841428)

[**Training Parameters** 21](#_Toc163841429)

[**Results:** 23](#_Toc163841430)

[**Potential Improvements to be Done:** 24](#_Toc163841431)

[Conclusion: 25](#_Toc163841432)

## **Project Overview:**

The project aims to develop a sentiment analysis system using LSTM and Seq2Seq models for text data. LSTM is a type of recurrent neural network (RNN) known for its ability to capture long-term dependencies in sequential data, making it suitable for NLP tasks like sentiment analysis. The Seq2Seq architecture, consisting of an encoder and decoder, is commonly used for tasks involving sequential data, such as machine translation and text summarization. In this project, we leverage LSTM for sentiment classification and Seq2Seq for text generation, with the goal of building a comprehensive NLP understanding on the 2 most common tasks of Natural Language Processing. For the purpose of learning and due to the lack of proper dataset, the models are implemented separately with the aim of demonstrating their function.

## **Covering Concepts:**

1. **Tokenization:** Tokenization is the process of breaking down text into smaller units, such as words or subwords, to facilitate further processing. It plays a crucial role in natural language processing tasks by converting raw text into manageable units that can be processed by machine learning models.
2. **Embedding:** Embedding is a technique used to represent words or phrases as dense vectors in a continuous vector space. These embeddings capture semantic relationships between words, allowing models to understand and generalize better from the text data. Embeddings are often learned during the training process of neural network models.
3. **RNN (Recurrent Neural Network):** RNN is a type of neural network architecture designed to handle sequential data by maintaining a state or memory across time steps. It is well-suited for tasks involving sequences, such as sentiment analysis and text generation. However, traditional RNNs suffer from the vanishing gradient problem, limiting their ability to capture long-term dependencies in sequences.
4. **LSTM (Long Short-Term Memory):** LSTM is a specialized type of RNN architecture developed to address the vanishing gradient problem and capture long-term dependencies in sequential data more effectively. It achieves this by incorporating a memory cell with gating mechanisms that regulate the flow of information, allowing it to retain important information over longer sequences.
5. **Sentiment Analysis**: Sentiment analysis is the task of classifying text into different sentiment categories (positive, negative, neutral) based on the emotions expressed. It is a fundamental task in natural language processing and has applications in various domains, including social media monitoring, customer feedback analysis, and market research.
6. **Seq2Seq (Sequence-to-Sequence):** Seq2Seq is an architecture consisting of an encoder and decoder neural network, commonly used for tasks involving sequential data, such as machine translation, summarization, and text generation. The encoder processes the input sequence and encodes it into a fixed-dimensional representation, while the decoder generates the output sequence based on this representation.
7. **Text Generation:** Text generation is the task of generating coherent and contextually relevant text based on a given input or prompt. It is often achieved using generative models, such as recurrent neural networks or transformers, which learn to predict the next word or token in a sequence based on the preceding context.

## **Potential Use Case:**

A potential use case for this sentiment analysis system lies in its application within social media sentiment monitoring tools or customer feedback analysis platforms. Consider a scenario where a company operates in a highly competitive market and seeks to maintain a strong brand reputation and customer satisfaction. In such a dynamic landscape, understanding public sentiment towards their products or services becomes paramount.

By integrating the LSTM-based sentiment analysis model into their existing infrastructure, the company can efficiently analyze vast amounts of textual data, including customer reviews, social media comments, and forum discussions. The model's ability to classify sentiment accurately allows the company to gain valuable insights into how customers perceive their offerings. For instance, they can identify positive sentiment indicating satisfaction with a recent product launch or detect negative sentiment signaling dissatisfaction with a particular feature.

Moreover, by incorporating the Seq2Seq model into their platform, the company can take sentiment analysis to the next level. Instead of solely analyzing sentiment, the system can generate responses or summaries with sentiment context. For instance, when responding to a negative review on social media, the system can generate a personalized message acknowledging the customer's concerns and offering solutions or assistance. Similarly, when analyzing customer feedback, the system can generate concise summaries highlighting prevailing sentiment trends or common themes.

This application empowers businesses to glean actionable insights from customer feedback, identify emerging trends, and address issues proactively. By leveraging sentiment analysis and response generation capabilities, the company can enhance customer engagement, foster positive brand sentiment, and ultimately improve customer loyalty and retention. Additionally, integrating the system into chatbots or customer service platforms enables the company to provide personalized responses in real-time, enhancing the overall customer experience and strengthening relationships with their audience. Overall, this use case underscores the transformative potential of sentiment analysis systems in enabling data-driven decision-making and customer-centric strategies in today's competitive market landscape.

## **Environment Used:**

Python programming language with:

1. os

2. yaml

3. keras = ^2.15.0

4. tensorflow = ^2.15.0

5. tensorflow\_text = ^2.15.0

6. numpy = ^1.26.3

## **Text Sentiment Classification:**

### **Dataset Used:**

For sentiment Analysis task, we will be using the IMDB dataset. The IMDB dataset is a widely-used benchmark dataset in natural language processing, containing movie reviews labeled with sentiment polarity. Each review is assigned a binary sentiment label indicating whether it expresses a positive or negative opinion. With its large collection of reviews, the IMDB dataset is valuable for training and evaluating machine learning models for sentiment analysis tasks.

A screenshot of a white box

Description automatically generated

Fig 1. Sample Data from IMDB Dataset

### **Preprocessing:**

For preprocessing, we will use common techniques in the industries to deal with text data. Neural Networks only accept integers as the inputs instead of strings, thus it is important for us to convert the text into machine readable integers. The preprocessing step consists of four main tasks: finding out the vocabulary and tokenizing the text data:

1. **Finding out the vocabulary:** This step involves extracting all unique words or tokens present in the text data. The vocabulary represents the entire set of words used in the dataset, excluding any duplicates. By identifying the vocabulary, we gain insights into the diversity and complexity of the language used in the dataset, which is crucial for building effective natural language processing models.
2. **Tokenizing the text data:** Tokenization is the process of breaking down the text data into smaller units, typically words or subwords, called tokens. This step involves splitting the text into individual tokens based on predefined rules, such as whitespace or punctuation. Tokenization enables further processing of the text data, such as numerical encoding or feature extraction, by converting raw text into a structured format that can be understood by machine learning algorithms.
3. **Numerical encoding:** After tokenization, each token in the text data needs to be represented numerically to be used in machine learning models. This step involves assigning a unique numerical identifier (in this case the index) to each token in the vocabulary. Numerical encoding allows the text data to be converted into a numerical format that can be processed by machine learning algorithms, facilitating model training and analysis.
4. **Padding with zeros:** In this step, the numerical sequences representing the text data are padded with zeros to ensure uniform length. Since neural network models require input sequences of equal length, we use padding to ensure that shorter sequences are extended with zeros to match the length of the longest sequence in the dataset. Padding with zeros ensures that all input sequences have the same shape, enabling efficient batch processing during model training without introducing bias.

### **Model Selection:**

A simple neural network consists of the following layers with different functions is implemented to achieve sentiment classification:

1. **Embedding Layer:**
   * Function: The embedding layer is responsible for converting input text data, typically represented as sequences of words or tokens, into dense vectors of fixed size. Each word or token in the input sequence is mapped to a unique dense vector representation, often learned during the training process.
   * Importance: The embedding layer captures semantic relationships between words and encodes contextual information into dense vectors. This allows the model to understand the meaning of words and their relationships within the input text data, facilitating effective sequence processing by subsequent layers.
2. **LSTM (Long Short-Term Memory) Layer:**
   * Function: The LSTM layer is a type of recurrent neural network (RNN) architecture designed to process sequential data while addressing the vanishing gradient problem and capturing long-term dependencies. It consists of memory cells with gated units that regulate the flow of information over time. The LSTM layer processes input sequences sequentially, updating its internal state based on the current input and previous state.
   * Importance: The LSTM layer is crucial for capturing temporal dependencies and contextual information within sequential data, such as text or time series. It allows the model to retain important information over long sequences, making it well-suited for tasks involving natural language processing, sentiment analysis, and sequence prediction.
3. **Dense Layer (Output Layer):**
   * Function: The dense layer, also known as the output layer, performs the final transformation of the learned features into the desired output format. It typically consists of one or more densely connected neurons, where each neuron is connected to every neuron in the previous layer. The dense layer applies a linear transformation followed by an activation function to produce the final output.
   * Importance: The dense layer is responsible for making predictions or generating output based on the learned features extracted from the input data by previous layers. In classification tasks, the dense layer often applies a softmax activation function to produce probability scores for each class label. In regression tasks, it may use a linear activation function to produce continuous output values.

A screen shot of a computer code

Description automatically generated

Fig 2. Code for Model Implementation

A diagram of a computer program

Description automatically generated

Fig 3. Model Structure Visualization

### **Training Parameter:**

In the model, the embedding layer is set to output vector embeddings of 32 dimensions, commonly used in practice and have been found to provide good performance across various natural language processing tasks.

1. **Embedding Dimension:** The embedding dimension refers to the size of the dense vector representations produced by the embedding layer. A higher embedding dimension allows the model to capture more nuanced semantic relationships between words, while a lower dimensionality may lead to more compact representations. In this case, the chosen embedding size of 32 strikes a balance between capturing meaningful semantic information and maintaining computational efficiency.
2. **Activation Function:** For output, a Sigmoid activation function is used. Using the sigmoid activation function on the output layer for binary classification tasks offers several advantages. Firstly, it transforms the output into probabilities, facilitating straightforward interpretation where values closer to 0 represent low confidence in the positive class, and values closer to 1 indicate high confidence. This probabilistic interpretation allows for easy thresholding, simplifying decision-making.

Moreover, the sigmoid function aligns well with the logistic loss function, optimizing model parameters to output accurate probabilities for binary classification. Additionally, its smooth derivative enables efficient gradient propagation during backpropagation, promoting stable training. Overall, the sigmoid activation function is a popular choice for binary classification problems due to its probabilistic nature, compatibility with loss functions, and support for gradient-based optimization.

1. **Optimizer Selection:** For optimizer, RMSprop is used instead of the most common Adam optimizer. RMSprop is an adaptive learning rate optimization algorithm that addresses the problem of vanishing or exploding gradients in training deep neural networks. It calculates an exponentially weighted moving average of squared gradients for each parameter and uses this moving average to adjust the learning rate for each parameter individually.

In the context of NLP text classification tasks, such as sentiment analysis or document categorization, RMSprop's advantages can be particularly beneficial. When dealing with textual data, the vocabulary size can be large, and the gradients can vary significantly across different words or features. RMSprop's adaptive learning rate mechanism allows it to adjust the learning rate for each parameter individually based on the magnitude of gradients. This adaptability helps stabilize the training process, prevent drastic changes in parameter updates, and promote smoother convergence, especially in scenarios where the gradients vary widely across different words or features. As a result, RMSprop can effectively optimize the parameters of neural networks for NLP text classification tasks, leading to more reliable and efficient training processes and potentially better classification performance.

1. **Training Callbacks:** Lastly, an early stopping callback of patience 3 is used to address overfitting as data augmentation for text data is harder to implement as compared to image data. Early stopping is a regularization technique that helps prevent overfitting and improve generalization by monitoring the model's performance on a validation dataset during training and halting training when performance begins to degrade. In the context of text classification, early stopping can offer several benefits. Firstly, it prevents the model from overfitting to the training data by halting training when the model starts to memorize noise or irrelevant patterns. This is particularly important in text classification tasks where the dataset may contain noisy or uninformative features. Additionally, early stopping helps save computational resources and training time by avoiding unnecessary epochs once the model has converged. By stopping training at the right moment, early stopping ensures that the model achieves optimal performance on unseen data while avoiding overfitting, making it a valuable tool for improving the effectiveness and efficiency of text classification models.

A screen shot of a computer code

Description automatically generated

Fig 4. Codes that Specify Training Hyperparameters and Activation Function in the model



Fig 5. Codes that Specify Loss and Optimizer Used.

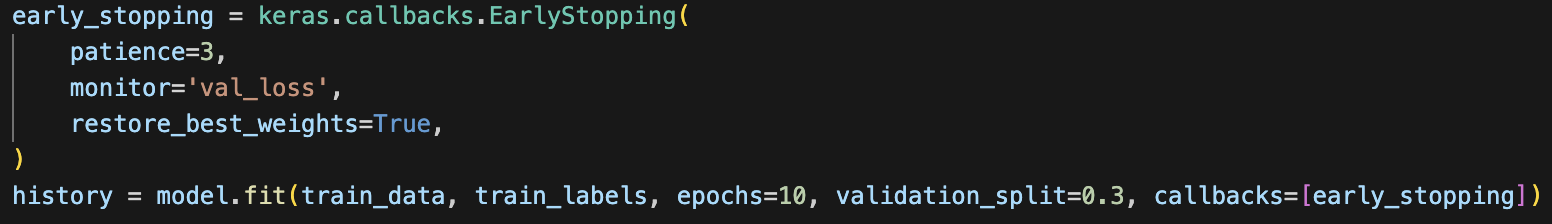


Fig 6. Codes that Specify Early Stopping Callbacks and Implement Training

### **Results:**

The model achieved a validation accuracy of 87.47%, surpassing the baseline. Testing on random strings typed results in correct classifications at around 12ms, suitable for mobile app integration. To test out the effect of embedding, sentences with different structures are used, some contains words that incurs drastic change of meaning. An example sentence is ” I thought the movie was going to be bad, but it was actually great!”, the model is able to correctly classify it in 12 ms. One limitation of the model made is that it is ineffective in classifying sentences that are overly short, for example like “It is bad”.

A screenshot of a computer

Description automatically generated

Fig 7. Model Testing with Online Photo

### **Potential Improvements to be Done:**

1. Finetune with different datasets. Explore other movie review datasets to expand the vocabulary and allows more accurate prediction.
2. Increase the dimension of the vector output from embedding layer, this will allow more information to be retained at the cost of computational resources.
3. Hyperparameter tuning: Further optimize model performance by adjusting parameters such as batch size, and optimizer choice.

## **Text Generation**

### **Dataset Used**

For text generation, we will be using the chatterbot dataset to create a mini chatbot.

The Chatterbot English dataset, available on Kaggle.com, comprises conversational data in English language. It consists of a collection of dialogues or exchanges between users and chatbots, covering various topics and conversational contexts. This dataset serves as a valuable resource for training and evaluating conversational AI models, such as chatbots or virtual assistants, allowing researchers and developers to build and improve natural language understanding and generation capabilities.

A white background with black text

Description automatically generated

Fig 8. Sample Data From the Chatterbot Dataset

### **Preprocessing:**

While still aligning with the common techniques for preprocessing, a few more steps are needed. For the text generation task using an encoder-decoder model, the preprocessing procedure involves several steps to prepare the data for training:

1. **Removing Short Sequences:**
   * Remove question and answer pairs that are too short to provide meaningful context for training. This ensures that the model learns from sufficiently informative input-output pairs.

A computer screen with text

Description automatically generated

Fig 9. Code for Removing Q&As that are too short

1. **Adding Start and End Tokens:**
   * Add special start and end tokens to each question and answer sequence. These tokens serve as delimiters to indicate the beginning and end of each sequence, providing clear boundaries for the model during training and generation.

A black background with blue and white text

Description automatically generated

Fig 10. Code for Adding “Start” and “End” tokens.

1. **Tokenizing and Padding for Questions:**
   * Tokenize each question sequence by breaking it down into individual words or subwords, creating a sequence of tokens. This step enables the model to process the text data as numerical inputs.
   * Pad the tokenized question sequences with zeros to ensure uniform length. Padding ensures that all input sequences have the same length, facilitating batch processing during training.

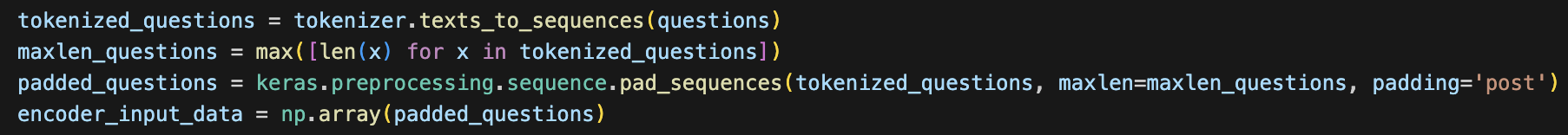


Fig 11. Tokenizing and padding All Questions

1. **Tokenizing and Padding for Answers:**
   * Tokenize each answer sequence similarly to the questions, creating a sequence of tokens.
   * Prepare a second tokenized sequence of answers that is offset by one position. This creates a teacher-forcing setup, where the model learns to predict the next token in the sequence based on the previous token.
   * Pad both sets of tokenized answer sequences with zeros to ensure uniform length, maintaining consistency with the padded question sequences.

A screen shot of a computer program

Description automatically generated

Fig 11. Tokenizing and padding All Answers Twice

### **Model Selection:**

The model selection involves two main components: the encoder and the decoder.

**Encoder:**

1. **Embedding Layer:**
   * **Function:** The embedding layer converts input text data, such as source language sentences in machine translation, into dense vectors of fixed size. Each word or token in the input sequence is mapped to a unique dense vector representation, facilitating semantic understanding.
   * **Importance:** The embedding layer captures semantic relationships between words and encodes contextual information into dense vectors, enabling the model to understand the meaning of words and their relationships within the input text data.
2. **LSTM (Long Short-Term Memory) Layer:**
   * **Function:** The LSTM layer processes the embedded input sequence sequentially, capturing temporal dependencies and contextual information within the data. It consists of memory cells with gated units that regulate the flow of information over time, allowing the model to retain important information over long sequences.
   * **Importance:** The LSTM layer is crucial for capturing long-range dependencies and contextual information within sequential data, such as sentences or sequences of words. It enables the model to encode the input sequence into a fixed-size context vector, summarizing the input information for further processing by the decoder.

**Decoder:**

1. **Embedding Layer:**
   * **Function:** Similar to the encoder, the embedding layer in the decoder converts target language sentences or output sequences into dense vectors of fixed size. Each word or token in the output sequence is mapped to a unique dense vector representation.
   * **Importance:** The embedding layer in the decoder allows the model to understand the meaning of words and their relationships within the target language sentences, facilitating the generation of coherent and meaningful translations or outputs.
2. **LSTM (Long Short-Term Memory) Layer:**
   * **Function:** The LSTM layer in the decoder processes the embedded target sequence sequentially, generating output tokens one at a time while considering the context vector produced by the encoder. It captures temporal dependencies and contextual information within the data, facilitating accurate sequence generation.
   * **Importance:** The LSTM layer in the decoder is crucial for generating coherent and contextually relevant output sequences based on the information encoded by the encoder. It allows the model to predict the next token in the sequence while considering the context of the input sequence, enabling accurate and fluent generation of translations or outputs.
3. **Dense Layer (Output Layer):**
   * **Function:** The dense layer in the decoder performs the final transformation of the learned features into the desired output format. It typically consists of one or more densely connected neurons, where each neuron is connected to every neuron in the previous layer. The dense layer applies a linear transformation followed by an activation function to produce the final output token probabilities.
   * **Importance:** The dense layer in the decoder is responsible for predicting the next token in the output sequence based on the context vector produced by the encoder and the previously generated tokens. It enables the model to generate accurate and contextually relevant output sequences in tasks such as machine translation or text generation.

A screen shot of a computer code

Description automatically generated

Fig 10. Code for Model Implementation

A diagram of a computer program

Description automatically generated with medium confidence

Fig 11. Encoder-Decoder Model Structure

### **Training Parameters**

The choice of training parameters for the encoder-decoder model training plays a critical role in the model's performance and convergence. Here's an explanation of the selected training parameters:

1. **Encoder Dimension of 200:**
   * The encoder dimension refers to the size of the hidden state or context vector produced by the LSTM layer in the encoder. A higher dimensionality allows the model to capture more complex patterns and information from the input sequences. In this case, the chosen encoder dimension of 200 provides a sufficiently large space for the model to encode and represent the input sequences effectively. This larger dimensionality enables the model to capture more intricate relationships and nuances within the input data, which is crucial for tasks like machine translation or text generation where preserving semantic meaning is essential.
2. **RMSprop Optimizer:**
   * RMSprop is selected as the optimizer for training the encoder-decoder model. RMSprop is an adaptive learning rate optimization algorithm that adjusts the learning rate for each parameter individually based on the magnitude of gradients. This adaptability is particularly advantageous in NLP tasks, where the gradients can vary significantly across different words or features in the vocabulary. By adjusting the learning rate dynamically, RMSprop helps stabilize the training process, prevent drastic changes in parameter updates, and promote smoother convergence. Additionally, RMSprop addresses the problem of vanishing or exploding gradients commonly encountered in training deep neural networks, making it a suitable choice for optimizing the parameters of the encoder-decoder model.
3. **Loss of Categorical Crossentropy:**
   * Categorical crossentropy is chosen as the loss function for training the encoder-decoder model. Categorical crossentropy is well-suited for multi-class classification tasks, where each target sequence represents a categorical distribution over multiple classes or tokens. In the context of the encoder-decoder model, the output sequences are represented as one-hot encoded vectors, and the model predicts the probability distribution over the vocabulary for each time step. Categorical crossentropy measures the dissimilarity between the predicted probability distribution and the true distribution of the target sequences, providing a suitable objective function for training the model to generate accurate and contextually relevant output sequences.

A screen shot of a computer code

Description automatically generated

Fig 12. Code Implementation for Training Hyperparameters

### **Results:**

The model is able to achieve categorical loss of 0.0670 after 150 episodes of training. Deploying the model to answer simple questions like “can you move” results in clear and understandable returns. However, as the vocabulary built for this dataset is small, questions asked beyond the vocabulary bank results in inaccurate return.

A screenshot of a computer program

Description automatically generated

Fig 13. Training Results

A computer screen shot of text

Description automatically generated

Fig 14. Model Testing

### **Potential Improvements to be Done:**

1. Finetune with different datasets. Explore other movie review datasets to expand the vocabulary and allows more accurate prediction.
2. Hyperparameter tuning: Further optimize model performance by adjusting parameters such as batch size, and optimizer choice.

## **Conclusion:**

n conclusion, this project delved into essential preprocessing techniques in NLP, covering word tokenizing and word embedding. It also provided insights into advanced concepts like Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Sequence-to-Sequence (Seq2Seq) models for text generation. RNNs and LSTMs excel at capturing temporal dependencies in sequences, while Seq2Seq models facilitate tasks like machine translation and text summarization. By exploring these advanced concepts, the project equipped learners with the skills to tackle complex NLP tasks effectively.

Top of Form

Bottom of Form