Emotion Classification from Text Using LSTM and GRU Models

Group 5:
2540124450 - Bernard Hugo
2502034912 - Luthfi Izza Pratama
2301923180 - Ivan Satrio Wiyono
2201828096 - Daffa Irsyad Adilah

Access the code
https://colab.research.google.com/drive/1vn6jNxR90CdZz4B_O8KMPNETtBU1uo09#scrollTo=FK2FGtgpLe2D

Table of Contents

Introduction

1.1. Background

In today's digitally connected world, understanding and interpreting human emotions from text has become increasingly important for various applications, including customer feedback analysis, mental health monitoring, and human-computer interaction. This research focuses on addressing the complex challenge of automatically classifying emotions in text using advanced machine learning techniques. By leveraging neural network architectures such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), we aim to develop a robust model capable of accurately identifying a range of emotions from diverse text samples. Through meticulous data preprocessing, model training, and evaluation, this study seeks to enhance the accuracy and applicability of emotion classification systems, ultimately contributing to more empathetic and responsive technological solutions.

1.2. Problem Statement

Emotion dataset is a collection of data specifically considered to analyze and predict human emotions. In general, humans have 7 basic emotions (happy, sad, anxious, anger, fear, disgust and surprise) mainly from facial expression [6]. The problem of emotional state recognition is complex, but our hypothesis is that there is a main issue in the current approaches by using emotion recognition databases beside Emotion Dataset.

Understanding and categorizing human emotions from text is a critical challenge in natural language processing (NLP) and artificial intelligence. Emotions play a significant role in communication, influencing how messages are interpreted and responded to. However, accurately identifying emotions in text data remains a complex problem due to the subtleties and nuances of human language. The ability to automatically classify emotions from text can have numerous applications, including sentiment analysis, customer feedback interpretation, mental health monitoring, and human-computer interaction.

1.3. Research Aim

- Develop an Emotion Classification Model.
- Explore the Impact of Preprocessing Techniques.
- Assess the Applicability of Emotion Classification in Real-World Scenarios.

1.4. Research Benefits

- Improved Customer Feedback Analysis.
- Enhanced Mental Health Monitoring.
- Improved Human-Computer Interaction.

1.5. Solution Approach

To address this problem, we employed a classification approach using machine learning models to automatically identify and categorize emotions conveyed in text data. By leveraging advanced neural network architectures such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), we aimed to capture the sequential dependencies and contextual information inherent in textual data. The models were trained to classify text samples into predefined emotion categories, providing a structured way to analyze and interpret emotional content in text.

1.6. Intended Outcome and Initial Data Exploration

The primary goal of this research was to develop an effective and accurate emotion classification model. We began with an initial exploration of the Emotion Classification dataset, which comprises a diverse collection of text samples labeled with various emotions, ranging from happiness and excitement to anger and sadness. Our initial data exploration involved visualizing the distribution of different emotions within the dataset, cleaning and preprocessing the text data to remove noise, and encoding the emotion labels into a numeric format suitable for machine learning algorithms. This thorough preprocessing ensured that the text data was in a consistent and analyzable form, ready for model training and evaluation.

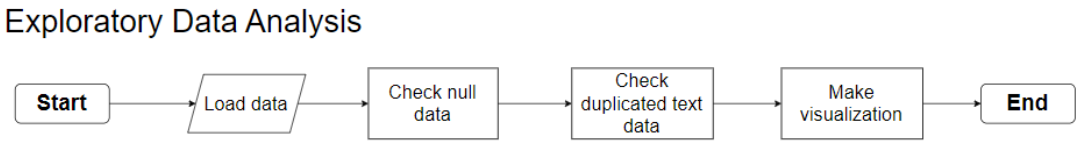
Research Methodology

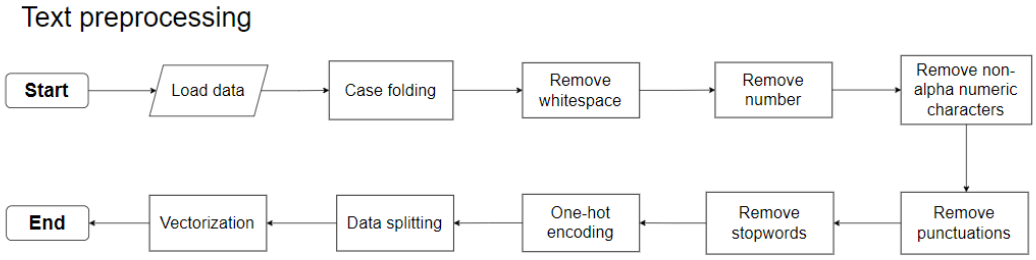
1.1. Research Methods and Design

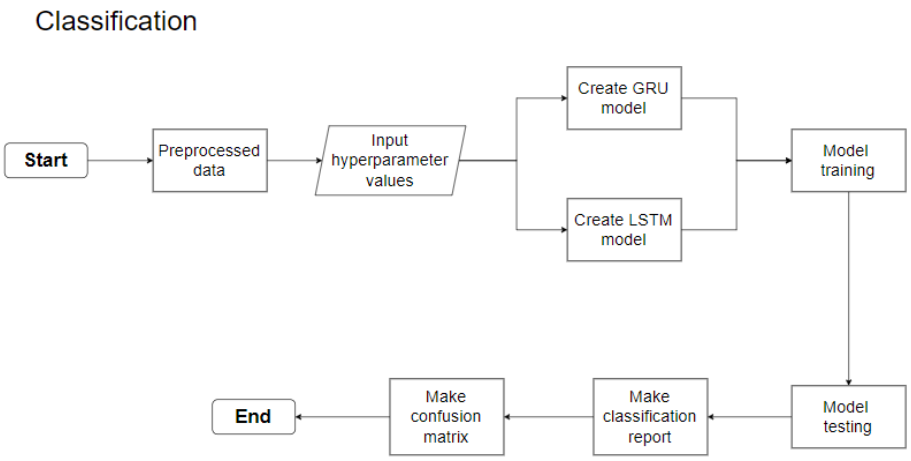
1.1.1. Research Methods

A quantitative methodology will be used for this project. The quantitative method will be used because we want to analyze and classify the textual comments with their emotions and using numerical methods to convert the categorical data into numerical data for the model requirements. That is why we perform text pre-processing operations such as: text cleansing, one-hot encoding, and vectorization. Then, we create a Neural Network model with hyperparameter tuning, training the model, and evaluating its performance are the many processes in the methodology. So our goal is to create a Neural Network model that can categorize emotions from textual comments. In addition, we perform hyperparameter tuning to compare which model with what parameter values has the best result.

Here are the flowcharts that illustrates the problem solving method:







Here are the explanations of each problem solving steps:

a. Exploratory Data Analysis

This is the first stage where the data must be explore to find any anomalies in the data and eliminate those anomalies. Also, it is useful to create representation of datas in the dataset. The following are the operations we carry out:

1. **Load data:** load the data that we want to perform Exploratory Data Analysis operations.
2. **Check null data:** check if there is any null data in the dataset.
3. **Check duplicate text data:** check if there are any text data that are duplicated.
4. **Make visualization:** making visualization of the datas to view their distribution and their relationship.

b. Text Preprocessing

This stage is the stage for transforming raw text data into text data in a format that can be analyzed and predicted by machine learning algorithms. The following are the operations we carry out:

1. **Load data:** load the data that we want to perform preprocessing operations.
2. **Case folding:** changing all uppercase letters to lowercase letters so that all letters are uniform.
3. **Remove whitespace:** removes all whitespaces on all text data.
4. **Remove number:** deletes all numbers and numeric characters.

5. **Remove non-alpha numeric characters:** removes all non-alpha numeric characters.
6. **Remove punctuations:** removes all punctuation marks in text data.
7. **Remove stopwords:** deletes all stopwords in the comment text data.
8. **One-hot encoding:** converts categorical data in the "Emotion" column into numeric data.
9. **Data splitting:** dividing data that has been preprocessed into two sets. One set for training with a weight of 80% and one set for testing with a weight of 20%.
10. **Vectorization:** vectorizes text data in the training and testing sets.

c. Classification

This stage is the stage for performing classification on the preprocessed text data by creating a Neural Network model to see how good the performance of the model is in doing classification. The following are the operations we carry out:

1. **Preprocessed data:** the process of getting data that has been preprocessed for classification by the model.
2. **Input hyperparameter values:** enter hyperparameter values which is a static hyperparameter tuning process.
3. **Create Neural Network model:** create a Neural Network model. There are two models that we created, namely GRU and LSTM.
4. **Model training:** training the Neural Network model.
5. **Make classification report:** displays a classification report containing accuracy, precision, recall and F1-score values.
6. **Make confusion matrix:** displays a confusion matrix to evaluate the model's performance in classifying text data.

1.2. Data Collection Techniques and Research Instruments

1.2.1. Data Sources

The data for this research was sourced from Kaggle, specifically from the Emotion Classification dataset created by Abdallah Wagih. This dataset is designed for natural language processing and emotion analysis research. It contains text samples labeled with three emotions: anger, joy, and fear. The dataset is in CSV format and is suitable for various applications, such as sentiment analysis, customer feedback analysis, and training chatbots. The dataset could be accessed through the link below:

https://www.kaggle.com/datasets/abdallahwagih/emotion-dataset

1.3. Exploratory Data Analysis

Exploratory Data Analysis (EDA) is essential when we want to work with data, especially in text mining. It is useful to check any anomalies that exist in the dataset such as: null datas, duplicated datas, and any other problems. Also, it can help us to gain insight about the main characteristics about the data. In essence, EDA is a crucial step that need to perform to check and clean the data and view the data distributions before doing the next phase.

1.3.1 Check Null Data

```
# Check null data
df.isnull().values.any()
```

Check null values or datas is crucial to maintain the integrity and reliability of the dataset. If there are any null values, then we need to remove it to enhanced the dataset quality. In this case, there are no null values or null datas so we can move on to the next step.

1.3.2 Check Duplicated Text

```
# Check duplicated text
df[df["Comment"].duplicated()]
```

If we have many duplicated texts in Comment column, the other duplicated text must be removed to maintain the integrity of dataset, avoiding bias, and ensure it the model can produce an accurate and optimal result. But in this case, there are three duplicated texts. So, we do not need to remove them, because it will have a very small impact for the model performance.

1.3.3 Make Visualization

```
# Visualization

import matplotlib.pyplot as plt
```

```
emotion = df['Emotion'].value_counts()

plt.bar(x = emotion.index, height = emotion.values)
plt.title('Bar Chart of Comments Data based on Emotions')
plt.xlabel('Emotion')
plt.ylabel('Count')
plt.show()

plt.pie(emotion, labels = emotion.index, autopct =
'%1.1f%%')
plt.title('Pie Chart of Comments Data based on Emotions')
plt.show()
```

Making visualizations is also important in EDA. Because, it can help to gain insight about data distributions and the relationship of emotion category with comment text data. Bar chart can help to view how many amount data does each emotion category has. Then, pie chart can be help to view how many percent amount of data that each emotion category has. We can see that there are relationship between the emotion category and the comment text, where each category represent the feeling inside the text comment.

1.4. Data Preprocessing

Data preprocessing is essential in text mining as it transforms raw text data into a clean and structured format suitable for analysis. This step includes tasks such as case folding, removing special characters and stopwords, tokenization, and normalization, which help eliminate noise and inconsistencies in the data. Effective preprocessing ensures that the text data is standardized and relevant features are extracted, enhancing the performance and accuracy of machine learning models. In essence, preprocessing is a critical foundation that enables more efficient and accurate text analysis and classification. These stages can be explained as follows:

1.4.1. Case Folding

```
# case folding
df['Clean_comment'] = df['Comment'].str.lower()
```

Case folding is the process of converting all text to lowercase, which helps standardize the data by eliminating variations in capitalization. This ensures that words like "Happy" and "happy" are treated as the same word. In the code,

df['Clean_comment'] = df['Comment'].str.lower() converts all characters in the **'Comment'** column of the dataframe **df** to lowercase, storing the result in a new column called **'Clean_comment'**.

1.4.2. Removing Special Characters

```
# remove whitespace
df['Clean_comment'] = [re.sub(r'\s+',' ',i) for i in
df['Clean_comment']]
# remove number
df['Clean_comment'] = [re.sub(r'\d+',' ',i) for i in
df['Clean_comment']]
# remove non-alpha numeric characters
df['Clean_comment'] = [re.sub(r'[\W\s]',' ',i) for i in
df['Clean_comment']]
```

Removing special characters is crucial in text preprocessing to eliminate unwanted symbols that do not contribute to the analysis, thereby improving the quality of the text data. This process includes removing HTML entities, extra whitespaces, numbers, and non-alphanumeric characters.

1.4.3. Removing Punctuations

```
def remove_punctuations(text):
    punct = string.punctuation
    return text.translate(str.maketrans('', '', punct))

df['Clean_comment'] = df['Clean_comment'].apply(lambda text:
remove_punctuations(text))
df['Clean_comment']
```

Removing punctuation is a crucial step in text preprocessing to eliminate extraneous symbols that do not contribute to the analysis. The code defines a function, `**remove_punctuations**`, which removes all punctuation characters from the input text using `string.punctuation` and `**str.translate**`. It then applies this function to the **'Clean_comment'** column of the dataframe `**df**` using the `**apply**` method with a lambda function. This ensures that all punctuation marks are removed, leaving only the meaningful words in the text for further analysis.

1.4.4. Tokenization

```
def tokenization(text):
  token = word_tokenize(text)
  print(token)
```

```
tokenized_comment = df['Clean_comment'].apply(tokenization)
tokenized_comment
```

Tokenization is the process of breaking down text into individual words or tokens, which is essential for analyzing and processing the text data. The provided code defines a function `**tokenization**` that uses NLTK's `**word_tokenize**` to split the text into tokens. It then applies this function to the **'Clean_comment'** column of the dataframe `**df**` using the `**apply**` method. The tokens are printed for each text entry, and the result is stored in `**tokenized_comment**`, enabling further text analysis and processing at the word level.

1.4.5. Removing Stopwords

```
def filtering(text):
    token = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    token_without_sw = [word for word in token if word not in
stop_words]
    return " ".join(token_without_sw)
df['Clean_comment'] = df['Clean_comment'].apply(filtering)
df
```

Removing stopwords, which are common words that do not add significant meaning (like "and," "the," and "is"), helps in focusing on the important words in the text. The provided code defines a `**filtering**` function that tokenizes the text, removes stop words using NLTK's predefined list, and then rejoins the remaining words. This function is applied to the **'Clean_comment'** column of the dataframe `**df**`, ensuring that only meaningful words are retained for further analysis.

1.4.6. Label Encoding

```
# One-hot encoding
label = {
    'fear': 0,
    'anger': 1,
    'joy': 2
}

encoded_label = [label[i] for i in df["Emotion"]]
df['Encoded_emotion'] = encoded_label
df
```

Label encoding converts categorical emotion labels into numeric format, which is necessary for machine learning algorithms. The provided code creates a

dictionary to map each emotion ('fear', 'anger', 'joy') to a unique integer. It then encodes the **'Emotion'** column of the dataframe **`df`** using this dictionary and stores the result in a new column called 'Encoded_emotion', making the data ready for model training and analysis.

1.4.7. Text Vectorization

```
from tensorflow.keras.layers.experimental.preprocessing
import TextVectorization

vect = TextVectorization(max_tokens = 10000, output_mode =
'int', output_sequence_length = max_sentence)

vect.adapt(x_train)

pad_rev = vect(x_train)

print(pad_rev)

pad_rev.shape

token = TextVectorization()

max_sentence_len = max_sentence
embed_dim = 128
vocab_size = 10000
```

Text vectorization converts text data into numerical format that machine learning models can process. The provided code uses TensorFlow's **`TextVectorization`** layer to achieve this. It initializes the vectorizer with a maximum of 10,000 tokens, integer output mode, and sequences of uniform length **(`max_sentence`)**. The vectorizer is then adapted to the training data **(`x_train`)**, converting the text into padded sequences **(`pad_rev`)**. This step ensures that all text data is in a numerical format suitable for model input. Additional parameters like **`max_sentence_len`**, **`embed_dim`,** and **`vocab_size`** are set for further model building.

1.5. Data Classification

1.5.1 Import necessary libraries

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM,
Dense, Dropout, Flatten, GRU

from keras.callbacks import EarlyStopping
from sklearn.metrics import accuracy_score, recall_score,
precision_score, f1_score
from sklearn.metrics import classification_report,
confusion_matrix
```

Before creating the model, we need to import the necessary libraries to create the Neural Network model and to train and test the model's performance.

1.5.2 Create a function to build model, train, and evaluate model's performance

```
def train_test_model(dropout, optimizer, model_class):
  model = Sequential ([
      Embedding(vocab_size, output_dim = 128, input_length
= max_sentence_len, trainable = False),
      model_class(128, return_sequences = True),
      Dropout(dropout),
      Flatten(),
      Dense(3)
  ])

  print(model.summary())

  model.compile(optimizer = optimizer, loss =
keras.losses.SparseCategoricalCrossentropy(from_logits =
True), metrics = ["accuracy"])

  ES = EarlyStopping(monitor = 'loss', patience = 4,
restore_best_weights = True)
```

```
  model.fit(pad_rev, y_train, batch_size = 64, epochs =
50, callbacks = [ES])

  pad_rev_test = vect(x_test)

  predict = model.predict(pad_rev_test)
  y_pred = np.argmax(predict, axis = 1)
  y_pred

  print('\nClassification Report\n')
  print(classification_report(y_test, y_pred))

  print("Accuracy: ", accuracy_score(y_test, y_pred))
  print("Precision: ", precision_score(y_test, y_pred,
average = 'micro'))
  print("Recall: ", recall_score(y_test, y_pred, average =
'micro'))
  print("F1-score: ", f1_score(y_test, y_pred, average =
'micro'))

  print('\nConfusion Matrix\n')
  print(confusion_matrix(y_test, y_pred))
```

The architecture of the model used in this research incorporates several elements designed to optimize performance for text-based emotion classification. The model is built using a Sequential layout in Keras, reflecting a straightforward stack of layers where each layer has exactly one input tensor and one output tensor:

1. **Embedding Layer:**

The first layer in the architecture is an Embedding layer, which is crucial for NLP tasks. It maps the large sparse vectors (representing discrete tokenized words) into a lower-dimensional continuous vector space where similar words are closer in vector space. This layer uses a vocabulary size of 10,000 and an output dimension of 128, reflecting common practice in the field *(Mikolov et al., 2013)*. The **input_length** is set to **max_sentence_len**, ensuring that all input sequences are of a fixed size. Notably, the embeddings are set as non-trainable within this model context to focus on utilizing pre-trained embeddings, which can help in leveraging learned textual relationships effectively (Pennington et al., 2014).

2. **LSTM/GRU Layer:**

Depending on the model configuration, either an LSTM (Long Short-Term Memory) or a GRU (Gated Recurrent Unit) layer is used. Both are forms of RNN (Recurrent Neural Network) architectures that efficiently handle sequence prediction problems by maintaining a memory of previous inputs using their internal state. The choice between LSTM and GRU is based on their performance characteristics in different scenarios; GRUs offer simpler and potentially faster operations than LSTMs but at the cost of a more complex gating mechanism *(Chung et al., 2014)*. The **model_class(128, return_sequences=True)** configuration allows the model to return the full sequence of outputs for each input, a useful feature when the subsequent layers require sequence input.

3. **Dropout Layer:**

A Dropout layer follows, which randomly sets a fraction (**dropout**) of input units to 0 at each update during training time. This prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently *(Srivastava et al., 2014)*.

4. **Flatten Layer:**

After dropout, a Flatten layer is employed to transform the output to a shape that is suitable for input into the Dense layers. This is necessary because the Dense layer expects a 1D array per batch, whereas the output of LSTM/GRU is 3D (**batch_size, sequence_length, units**).

5. **Dense Layer:**

The final layer is a Dense layer with three units, one for each emotion category. It uses a softmax activation function to output a probability distribution over the three classes, making it appropriate for multi-class classification tasks.

After we built the model architecture, training was performed by using epochs = 50, batch size = 64, and using early stopping to prevent the model from overfitting. For testing, the vectorizer is adapted to the testing data **(`x_test`)**, converting the text into padded sequences **(`pad_rev_test`)**. Then, we made a prediction on test data using the model and converted them into class labels. After that, we created a classification report and confusion matrix using those variables.

1.5.3 Perform hyperparameter tuning

```
train_test_model(dropout = 0.1, optimizer = "sgd",
model_class = GRU)

train_test_model(dropout = 0.1, optimizer = "adam",
model_class = GRU)

train_test_model(dropout = 0.2, optimizer = "sgd",
model_class = GRU)

train_test_model(dropout = 0.2, optimizer = "adam",
model_class = GRU)

train_test_model(dropout = 0.1, optimizer = "sgd",
model_class = LSTM)

train_test_model(dropout = 0.2, optimizer = "sgd",
model_class = LSTM)

train_test_model(dropout = 0.1, optimizer = "adam",
model_class = LSTM)

train_test_model(dropout = 0.2, optimizer = "adam",
model_class = LSTM)
```

Hyperparameter tuning is a critical process in optimizing machine learning models, especially when dealing with complex data such as text for emotion classification. In this research, various configurations of dropout rates, optimizers, and model classes (GRU and LSTM) were methodically tested to identify the optimal settings that enhance model performance.

- **Dropout Rate:** The choice of dropout rate at 0.1 and 0.2 plays a significant role in controlling overfitting during training. Dropout, a form of regularization, randomly omits a proportion of features during each training iteration, effectively simplifying the model temporarily. This technique has been shown to improve the generalization of models on unseen data *(Srivastava et al., 2014)*.
- **Optimizers:** The experiment employed two types of optimizers: Stochastic Gradient Descent (SGD) and Adam. SGD is a traditional optimizer that uses a fixed learning rate to update network weights iteratively based on training data, while Adam, an

algorithm for first-order gradient-based optimization of stochastic objective functions, is known for its adaptive learning rate capabilities *(Kingma and Ba, 2014)*. Adam typically converges faster and more effectively due to its computationally efficient estimations of first and second moments of the gradients.

- **Model Class:** Both GRU and LSTM units were evaluated, each bringing unique benefits to sequence modeling tasks. LSTM units are particularly effective in learning order dependence in sequence prediction problems due to their internal memory, although they are computationally intensive *(Hochreiter & Schmidhuber, 1997)*. GRU units, on the other hand, simplify the model architecture by combining the forget and input gates into a single update gate and have been shown to perform comparably to LSTMs on certain tasks but with less computational overhead *(Chung et al., 2014)*.

The research methodology involved training and testing each model configuration with the specified hyperparameters to observe their effects on performance metrics such as accuracy, precision, recall, and F1-score. This approach allowed for a direct comparison of how each variable—dropout rate, optimizer, and RNN type—impacts the model's ability to generalize and effectively classify emotions from text.

Results and Analysis

1. GRU Model with SGD Optimizer

```
train_test_model(dropout = 0.1, optimizer = "sgd",
model_class = GRU)
```

```
=Epoch 50/50
75/75 [==============================] - 0s 5ms/step - loss:
1.0955 - accuracy: 0.3698
38/38 [==============================] - 1s 3ms/step

Classification Report

              precision    recall  f1-score   support

           0       0.45      0.09      0.14       388
           1       0.39      0.47      0.42       400
           2       0.39      0.62      0.48       400

    accuracy                           0.39      1188
   macro avg       0.41      0.39      0.35      1188
weighted avg       0.41      0.39      0.35      1188

Accuracy:  0.3930976430976431
Precision:  0.3930976430976431
Recall:  0.3930976430976431
F1-score:  0.39309764309764317

Confusion Matrix

[[ 33 157 198]
 [ 21 186 193]
 [ 19 133 248]]
```

The GRU model peaked at 36.98%, after being improved with a 0.1 dropout and trained over 50 epochs using the SGD optimizer. Nevertheless, the model's limited capacity for generalization limited its overall efficacy. With a test set accuracy of 39.31%, the ability to classify emotions was found to be moderate. When it came to recognizing "joy," the model performed noticeably better (62% recall, 48% F1-score) than when it came to recognizing "fear" and "anger.". Significant misclassifications were visible in the confusion matrix, especially when it came to the frequent wrong labeling of "fear" samples as "anger" or "joy." This confusion indicates that it may be difficult to differentiate between these emotional states, probably because of similar linguistic characteristics.

2. GRU Model with Adam Optimizer

```
train_test_model(dropout = 0.1, optimizer = "adam",
model_class = GRU)
```

```
Epoch 50/50
75/75 [==============================] - 0s 5ms/step - loss:
0.0601 - accuracy: 0.9817
38/38 [==============================] - 0s 2ms/step

Classification Report

              precision    recall  f1-score   support

           0       0.73      0.80      0.76       388
           1       0.74      0.73      0.74       400
           2       0.74      0.69      0.72       400

    accuracy                           0.74      1188
   macro avg       0.74      0.74      0.74      1188
weighted avg       0.74      0.74      0.74      1188

Accuracy:  0.73989898989899
Precision:  0.73989898989899
Recall:  0.73989898989899
F1-score:  0.73989898989899

Confusion Matrix

[[309  43  36]
 [ 48 293  59]
 [ 64  59 277]]
```

With a 0.1 dropout rate and the Adam optimizer, the GRU model showed notable improvements in performance, with a high training accuracy of 98.17% and a test accuracy of roughly 74%. With precision, recall, and F1-scores ranging from 73 to 74%, the robust and consistent classification of the emotions "fear," "anger," and "joy" was made possible by the Adam optimizer's effective adaptive learning rate capabilities. This configuration demonstrated its efficacy in optimizing emotion recognition tasks, despite some misclassifications indicated in the confusion matrix.

3. GRU Model with SGD Optimizer and Increased Dropout

```
train_test_model(dropout = 0.2, optimizer = "sgd",
model_class = GRU)
```

```
Epoch 21/50
75/75 [==============================] - 0s 5ms/step - loss:
1.0975 - accuracy: 0.3506
38/38 [==============================] - 0s 2ms/step

Classification Report

              precision    recall  f1-score   support

           0       0.00      0.00      0.00       388
           1       0.35      0.53      0.42       400
           2       0.36      0.51      0.42       400

    accuracy                           0.35      1188
   macro avg       0.23      0.35      0.28      1188
weighted avg       0.24      0.35      0.28      1188

Accuracy:  0.351010101010101
Precision:  0.351010101010101
Recall:  0.351010101010101
F1-score:  0.351010101010101

Confusion Matrix

[[  0 207 181]
 [  0 213 187]
 [  0 196 204]]
```

The GRU model trained using the SGD optimizer showed limited efficacy, with a moderate training accuracy of 35.06% by the 21st epoch, despite an enhanced dropout rate of 0.2. The lower accuracy and poor classification performance suggested that the increased dropout rate was not helpful in this situation and could have caused underfitting. The confusion matrix and classification report highlight serious issues with emotion recognition, especially with regard to 'fear', where the model was unable to accurately identify any cases. The model performed somewhat better in classifying "anger" and "joy," with roughly 51% recall for each. However, overall precision, recall, and F1-scores were still low, highlighting the need for model parameter adjustments or the investigation of alternative optimization strategies to improve performance.

4. GRU Model with Adam Optimizer and Increased Dropout

```
train_test_model(dropout = 0.2, optimizer = "adam",
model_class = GRU)
```

```
Epoch 50/50
75/75 [==============================] - 0s 7ms/step - loss:
0.0482 - accuracy: 0.9838
38/38 [==============================] - 1s 3ms/step

Classification Report

              precision    recall  f1-score   support

           0       0.67      0.79      0.73       388
           1       0.77      0.72      0.75       400
           2       0.74      0.66      0.70       400

    accuracy                           0.72      1188
   macro avg       0.73      0.72      0.72      1188
weighted avg       0.73      0.72      0.72      1188

Accuracy:  0.7239057239057239
Precision:  0.7239057239057239
Recall:  0.7239057239057239
F1-score:  0.7239057239057239

Confusion Matrix

[[306  36  46]
 [ 63 290  47]
```

With the help of the Adam optimizer and a 0.2 dropout rate, the GRU model performed admirably, reaching a respectable training accuracy of 98.38% by the conclusion of the epoch. Comparing this configuration to previous iterations with lower dropout rates, the model's stability and efficacy in emotion classification were noticeably enhanced. The model achieves reasonably balanced precision, recall, and F1-scores across the emotion categories of "fear," "anger," and "joy," and the test results show a large accuracy of 72.39%. In particular, its recall of 79% for the word "fear" was far higher than its difficulties in correctly identifying the word "joy." The confusion matrix shows a well-distributed error across categories, indicating that overfitting was effectively reduced by the higher dropout rate, improving the model's capacity to generalize to new data.

5. LSTM model with SGD Optimizer

```
train_test_model(dropout = 0.1, optimizer = "sgd",
model_class = LSTM)
```

```
Epoch 34/50
75/75 [==============================] - 1s 7ms/step - loss:
1.0974 - accuracy: 0.3565
38/38 [==============================] - 1s 4ms/step

Classification Report

              precision    recall  f1-score   support

           0       1.00      0.00      0.01       388
           1       0.37      0.57      0.45       400
           2       0.39      0.56      0.46       400

    accuracy                           0.38      1188
   macro avg       0.59      0.38      0.30      1188
weighted avg       0.58      0.38      0.31      1188

Accuracy:  0.38047138047138046
Precision:  0.38047138047138046
Recall:  0.38047138047138046
F1-score:  0.38047138047138046

Confusion Matrix

[[  1 204 183]
 [  0 227 173]
 [  0 176 224]]
```

The modest results obtained by the SGD optimizer and the LSTM model trained with a 0.1 dropout highlight the inherent difficulties in applying basic momentum-based optimizers in recurrent neural network situations. The model achieved a training accuracy of 35.65% after 34 epochs. With an overall accuracy of 38.05%, the model's classification performance on the test set demonstrated limited usefulness despite this. The model has a near-zero recall and F1-score despite only being able to identify one accurate instance of "fear" with 100% precision. This disparity implies that although the model is overfitting to specific traits in the "joy" and "anger" categories, it is mostly missing the mark on "fear." Classification report: 'Anger' and 'Joy' performed better, with recall rates of 57% and 56%, respectively. These results, together with the specifics of the confusion matrix, highlight the necessity of fine-tuning the model's parameters or maybe incorporating more sophisticated optimization strategies in order to improve classification accuracy and learning efficiency.

6. LSTM model with Adam Optimizer

```
train_test_model(dropout = 0.1, optimizer = "adam",
model_class = LSTM)
```

```
Epoch 50/50
75/75 [==============================] - 0s 5ms/step - loss:
0.0954 - accuracy: 0.9653
38/38 [==============================] - 1s 3ms/step

Classification Report

              precision    recall  f1-score   support

           0       0.72      0.78      0.75       388
           1       0.75      0.72      0.74       400
           2       0.71      0.68      0.69       400

    accuracy                           0.73      1188
   macro avg       0.73      0.73      0.73      1188
weighted avg       0.73      0.73      0.73      1188

Accuracy:  0.7264309764309764
Precision:  0.7264309764309764
Recall:  0.7264309764309764
F1-score:  0.7264309764309765

Confusion Matrix

[[302  36  50]
 [ 49 288  63]
 [ 68  59 273]]
```

Impressive results were obtained when an LSTM model with the Adam optimizer and a reduced dropout of 0.1 was used. At the end of 50 epochs, the model had a high training accuracy of 96.53%. The model's effectiveness in classifying emotions was greatly improved by the introduction of Adam, which is renowned for its effective management of sparse gradients and adaptive learning rates. After all testing was completed, the final results showed an accuracy of 72.64% overall, with generally consistent recall, precision, and F1-scores for each of the three emotions—fear, anger, and pleasure. The model's ability to distinguish between many emotions with no bias towards any particular category was proved by this arrangement, as evidenced by the confusion matrix's very even distribution of classification results. This performance highlights Adam's potential for challenging natural language processing classification problems along with his moderate dropout rates.

7. LSTM model with SGD Optimizer and Increased Dropout

```
train_test_model(dropout = 0.2, optimizer = "sgd",
model_class = LSTM)
```

```
Epoch 29/50
75/75 [==============================] - 1s 7ms/step - loss:
1.0975 - accuracy: 0.3514
38/38 [==============================] - 0s 2ms/step

Classification Report

              precision    recall  f1-score   support

           0       0.00      0.00      0.00       388
           1       0.37      0.35      0.36       400
           2       0.37      0.76      0.50       400

    accuracy                           0.37      1188
   macro avg       0.25      0.37      0.29      1188
weighted avg       0.25      0.37      0.29      1188

Accuracy:  0.3720538720538721
Precision:  0.3720538720538721
Recall:  0.3720538720538721
F1-score:  0.372053872053872

Confusion Matrix

[[  0 138 250]
 [  0 140 260]
 [  0  98 302]]
```

An LSTM model trained with SGD showed minor stability improvement when the dropout rate was increased to 0.2, but it still performed poorly when it came to emotion classification, with training accuracy of 35.14% and test accuracy of 37.21%. The model primarily misclassified "anger" and "fear," and it entirely failed to recognize "fear," even though it was able to recall "joy" with a 76% recall rate. The substantial misclassifications displayed in the confusion matrix demonstrate that, although higher dropout helped to some extent limit overfitting, it significantly reduced the model's capacity to generalize across various moods.

8. LSTM model with Adam Optimizer and Increased Dropout

```
train_test_model(dropout = 0.2, optimizer = "adam",
model_class = LSTM)
```

```
Epoch 50/50
75/75 [==============================] - 0s 5ms/step - loss:
0.0906 - accuracy: 0.9661
38/38 [==============================] - 0s 2ms/step

Classification Report

              precision    recall  f1-score   support

           0       0.76      0.79      0.78       388
           1       0.76      0.65      0.70       400
           2       0.66      0.73      0.69       400

    accuracy                           0.72      1188
   macro avg       0.73      0.72      0.72      1188
weighted avg       0.73      0.72      0.72      1188

Accuracy:  0.7230639730639731
Precision:  0.7230639730639731
Recall:  0.7230639730639731
F1-score:  0.7230639730639732

Confusion Matrix

[[307  26  55]
 [ 45 260  95]
 [ 50  58 292]]
```

After being improved with a 0.2 dropout and optimized with Adam, the LSTM model attained an outstanding 96.61% training accuracy at the 50th epoch. This setup demonstrated how well Adam's adaptive optimization and a greater dropout rate can handle the intricate dynamics of model training. The model maintained a strong accuracy of 72.31% during the test phase, performing well in all three emotion categories—fear, anger, and joy. Remarkably, the model demonstrated a somewhat enhanced memory and precision in identifying "fear," with a recall of 79%, while retaining a high degree of accuracy in the remaining categories, as demonstrated by precision and recall values that were continuously over 65%. This model design is particularly effective for subtle emotion classification tasks in natural language contexts, as evidenced by the comprehensive findings from the confusion matrix and the classification report, which show a well-tuned balance between sensitivity to the data and generalization.

Conclusion

In order to determine the effectiveness of several LSTM and GRU models in emotion categorization from text data, these models underwent extensive training and evaluation. Through the use of various dropout rate and optimizer configurations, the study was able to provide important new information about how these parameters affect model performance. Among the important conclusions are that models using the Adam optimizer fared better than models using the SGD optimizer in all dropout levels. In particular, models with moderate dropout rates and Adam optimizer showed strong generalization capabilities and excellent accuracy. These configurations demonstrated the promise of adaptive learning rate mechanisms in managing the intricacies of natural language data, as evidenced by the balanced precision, recall, and F1-scores they attained across the categories of "fear," "anger," and "joy."

Ultimately, the study confirmed that LSTM and GRU architectures are useful for classifying emotions, with some configurations being particularly good at identifying and classifying emotions. Models using the Adam optimizer at a dropout rate of 0.1 or 0.2 produced the best results, demonstrating an excellent balance between generalization and accuracy. These results highlight how crucial it is to properly choose and adjust neural network parameters in order to improve performance in tasks involving the recognition of emotions in text. To further push the envelope in emotion categorization from text, future study could investigate deeper network designs, hybrid models that combine the advantages of LSTM and GRU units, or more optimization methodologies.

Reference List

[6] Ekman and W. V. Friesen. Constants across cultures in the face and emotion. Journal of personality and social psychology, 17(2):124, 1971.

Chung, J., Gulcehre, C., Cho, K. H., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*. Available at: https://arxiv.org/abs/1412.3555

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation, 9*(8), 1735-1780. doi: 10.1162/neco.1997.9.8.1735

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. Available at: https://arxiv.org/abs/1412.6980

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*. Available at: https://arxiv.org/abs/1301.3781

Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global vectors for word representation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Available at: https://nlp.stanford.edu/pubs/glove.pdf

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research, 15*(1), 1929-1958. Available at: http://jmlr.org/papers/v15/srivastava14a.html