Disaster Identification from Tweets

September 14, 2025

1 Disaster Tweet Identification with Topic Modeling via Non-negative Matrix Factorization

Megan Arnold

September 14, 2025

2 Table of contents

3 Introduction

This project applies unsupervised learning to identify disaster-related content in social media posts. The dataset, sourced from Kaggle[1], contains 7613 tweets labeled as either disaster-related or not. Disasters include topic/themes; such as, floods, fires, earthquakes, and disease. Labels for the dataset exist and is fairly balanced, about 43% of the data is the positive class (disaster-related). I will be using Recurrent Neural Networks to classify the tweets. I will being comparing the LSTM and GRU architectures, along with 2 embedding models and a fine-tuned embedding layer on the GRU model.

The motivation for this project is to help enable the performance of real-world disaster response.

Quickly identifying emerging events, especially on platforms like Twitter, can help enable emergency response coordination and enhance situational awareness for first responders. By identifying the latent topics inherent in the dataset, we can cluster the tweets that correspond to various disasters types, even in the absense of labeled data.

The project i structured around the following steps: - **Exploratory data analysis** to determine class imbalance, assess quality and language patterns, determine preprocessing requirements, and help guide the vectorization and modeling approaches. - **Tweet preprocessing** to remove noise, such as URLs, emojis, and special characters. - **Feature Engineering** by applying vecotrization methods to the text data. - **Baseline model** development using a simple LSTM architecture to establish a performance benchmark. - **Hyperparameter tuning** to optimize the model architecture, including dropout, layers, and embedding models - **Model Evaluation** using classification metrics; such as, accuracy, precision, recall, and f1 score, based on post-hoc mapping of the original dataset labels. - **Analyze** the impact of the model architecture and embedding choices on performance. - **Discuss** the implications of the findings, including limitations and potential applications. - State **recommendations** for future work and improvements to the methodology.

This project demonstrates the ability for Recurrent Neural Networks to classify short, messy, and informatl datasets, like tweets and social media posts. The ideal goal for a model like this, would be to enable real-time disaster detection and response coordination following anonomolous tweet activity.

4 Importing data and libraries

Load dataset, libraries, and verify GPU availability

[ ]:
```
import os
import gc

os.environ["TF_FORCE_GPU_ALLOW_GROWTH"] = "true"
os.environ["LD_LIBRARY_PATH"] = (
    "/usr/local/cuda/lib64:/usr/lib/x86_64-linux-gnu:"
    + os.environ.get("LD_LIBRARY_PATH", "")
)
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "1"  # Show important logs
os.environ["CUDA_VISIBLE_DEVICES"] = "0"  # Force use of GPU 0
import tensorflow as tf

tf.keras.backend.clear_session()
gc.collect()
import keras as k


# Check for GPU and enable memory growth
gpus = tf.config.list_physical_devices("GPU")
if gpus:
    try:
        for gpu in gpus:
```

```python
            tf.config.experimental.set_memory_growth(gpu, True)
        print("GPU is available and memory growth is enabled.")
    except RuntimeError as e:
        print("RuntimeError during GPU setup:", e)
else:
    print("No GPU detected. Check your driver and CUDA installation.")
print(k.__version__)

print(tf.config.list_physical_devices("GPU"))
print(tf.__version__)
print("Num GPUs Available: ", len(tf.config.list_physical_devices("GPU")))
print(tf.config.list_physical_devices("GPU"))
```

```
2025-08-23 10:48:31.309619: I tensorflow/core/util/port.cc:113] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-08-23 10:48:31.667526: E
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2025-08-23 10:48:31.667643: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2025-08-23 10:48:31.725101: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2025-08-23 10:48:31.849847: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
2025-08-23 10:48:32.840965: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

```
2.15.0
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
2.15.0
Num GPUs Available:  1
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
2025-08-23 10:48:34.722842: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:887] could not open
file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
```

```
2025-08-23 10:48:34.917599: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:887] could not open
file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
2025-08-23 10:48:34.917646: I
external/local_xla/xla/stream_executor/cuda/cuda_executor.cc:887] could not open
file to read NUMA node: /sys/bus/pci/devices/0000:01:00.0/numa_node
Your kernel may have been built without NUMA support.
```

4.0.1 Helper Functions

[ ]:
```python
from collections import Counter
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import pickle
import random
import re
import seaborn as sns

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import (
    confusion_matrix,
    classification_report,
    accuracy_score,
    f1_score,
    precision_score,
    recall_score,
    ConfusionMatrixDisplay,
)
from sklearn.model_selection import train_test_split

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences


ROOT_DIR = os.getcwd()
DATA_DIR = os.path.join(ROOT_DIR, "data")
MODEL_DIR = os.path.join(ROOT_DIR, "models")

pd.set_option("display.max_colwidth", None)


# Helper Functions
def clean_tweet(text):
    """Clean the tweet text by removing unwanted elements.
```

```
    Args:
        text (str): The original tweet text.

    Returns:
        str: The cleaned tweet text.
    """
    # Remove emojis (unicode ranges covering emoticons, symbols, pictographs,␣
↪flags, etc.)
    emoji_pattern = re.compile(
        "["
        "\U0001f600-\U0001f64f"  # emoticons
        "\U0001f300-\U0001f5ff"  # symbols & pictographs
        "\U0001f680-\U0001f6ff"  # transport & map symbols
        "\U0001f1e0-\U0001f1ff"  # flags
        "\U00002500-\U00002bef"  # chinese characters
        "\U00002702-\U000027b0"
        "\U000024c2-\U0001f251"
        "\U0001f926-\U0001f937"
        "\U00010000-\U0010ffff"
        "\u200d"
        "\u2640-\u2642"
        "\u2600-\u2b55"
        "\u23cf"
        "\u23e9"
        "\u231a"
        "\ufe0f"
        "\u3030"
        "]+",
        flags=re.UNICODE,
    )

    text = emoji_pattern.sub(r"", text)  # remove emojis
    text = re.sub(r"\d+", "", text)  # remove numbers
    text = re.sub(r"http\S+", "", text)  # remove URLs
    text = re.sub(r"@\w+", "", text)  # remove mentions
    text = re.sub(r"#\w+", "", text)  # remove hashtags
    text = re.sub(r"[^A-Za-z0-9\s]+", "", text)  # remove punctuation/special␣
↪chars
    text = text.lower().strip()  # lowercase and strip
    return text


def get_test_predictions(model, tokenizer, max_length, texts, ids):
    # Get test predictions and probabilities given model

    sequences = tokenizer.texts_to_sequences(texts)
```

```python
    padded = pad_sequences(sequences, maxlen=max_length, padding="post")
    probabilities = model.predict(padded)
    predicted_classes = (probabilities.flatten() >= 0.5).astype("int")
    df_complete = pd.DataFrame(
        {
            "id": ids,
            "text": texts,
            "target": predicted_classes,
            "probability": probabilities.flatten(),
        }
    )

    return df_complete, df_complete[["id", "target"]]


def get_tp_fp_tn_fn_samples(df, num_samples=3):
    # Get samples of true positives, false positives, true negatives, and false␣
 ↪negatives
    tp_samples = df[(df["target"] == 1) & (df["actual"] == 1)].
 ↪sample(num_samples)
    fp_samples = df[(df["target"] == 1) & (df["actual"] == 0)].
 ↪sample(num_samples)
    tn_samples = df[(df["target"] == 0) & (df["actual"] == 0)].
 ↪sample(num_samples)
    fn_samples = df[(df["target"] == 0) & (df["actual"] == 1)].
 ↪sample(num_samples)

    return (
        tp_samples["text"],
        fp_samples["text"],
        tn_samples["text"],
        fn_samples["text"],
    )


def load_model_and_history(model_name):
    # Load model training history from CSV
    # load pickel history
    history = pickle.load(
        open(f"{DATA_DIR}/submissions/{model_name}_history.pkl", "rb")
    )
    model = tf.keras.models.load_model(f"{DATA_DIR}/submissions/
 ↪{model_name}_model.h5")
    return model, history_df
```

4.1 Import Data

[ ]:
```
# Load Data

df = pd.read_csv(os.path.join(DATA_DIR, "train.csv"))
df_test = pd.read_csv(os.path.join(DATA_DIR, "test.csv"))
df["cleaned_text"] = df["text"].apply(clean_tweet)
df_test["cleaned_text"] = df_test["text"].apply(clean_tweet)
```

5 Exploratory Data Analysis

5.1 Data overview

The dataset used in this project was from Kaggle's **"Disaster Tweets"** dataset. Each tweet is labeled as either disaster-related (1) or non-disaster-related (0). The dataset was collected using a **keyword matching process**, meaning the tweets were filtered based on the presence of certain keywords potentially related to disasters; however, it doesn't guarantee that the tweets are actually disaster-related.

Below are some key features of the dataset that will be taking into consideration in the methodology: - **Class Imbalance**: The dataset is imbalanced, with 43.0% of the data labeled as "disaster" (1) and the rest as "non-disaster" (0). - The dataset contains 7613 tweets, with 4342 labeled as **"non-disaster"** (0) and 3271 labeled as **"disaster"** (1). The classes are relatively balanced and an analysis using accuracy is sufficent for the validation test set. The final analysis will include the classification metrics like recall, precision, and f1 score. The final test set submission to Kaggle will be accuracy, which is why I'm focusing on that for the validation set.

- **Feature Description**::
  - **id**: unique identifier for each tweet
  - **keyword**: keyword from the tweet that was matched in the data mining process
  - **location**: location of the tweet
  - **text**: text of the tweet
  - **target**: label (1 for disaster, 0 for non-disaster)
- **Data Quality**:
  - The dataset is relatively clean, with no missing values in the target column. However, there are many missing values location columns, along with inconsistent naming conventions.
  - Dropped **keyword** because it was used in the data mining process and will be a redundant feature.
  - Dropped **location** because it is not-relevant to the analysis and has many missing values along with inconsistent naming conventions.
  - The **text** column will be the main feature used for vectorization and modeling. However, certain aspects will be **removed** from the text; such as:
    * URLs
    * Punctuation
    * Emojis
    * Special characters
    * Numbers

    * English stop words
    * Foreign characters and words Having a cleaned dataset will ensure the transformation with a tokenizer and embedding models is effective. By removing noise from the text, we can focus on the meaningful words and tokens, instead of attempting to vectorize foreign characters, emojis, and other non-informative text.

The cleaned text will be used for vectorization and topic modeling. In the exploratory data analysis section, additional insights will be gained from the data, including the distribution of the tweet lengths and distribution of word frequencies.

Below are high level statistics about the dataset, including the class imbalance, number of tweets, and 10 examples of tweets from the dataset. Further analysis will be conducted regarding the distribution of tweet lengths, word frequencies, and other relevant features.

[10]:
```
print("\n\nDataset Balance (Percent of positive samples):\n")
print(df["target"].mean())

print("\n\nCount of Positive and Negative Samples:\n")
print(df["target"].value_counts())

print("\n\nDF Information:\n")
print(df.info())

print("\n\nDF Sample:\n")
print(df[["target", "text"]].sample(10))

print("\n\nDF Location :\n")
print(df["location"].value_counts())

print("\n\nDF Keyword :\n")
print(df["keyword"].value_counts())
```

```


Dataset Balance (Percent of positive samples):

0.4296597924602653


Count of Positive and Negative Samples:

target
0    4342
1    3271
Name: count, dtype: int64


DF Information:

```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7613 entries, 0 to 7612
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   id            7613 non-null   int64
 1   keyword       7552 non-null   object
 2   location      5080 non-null   object
 3   text          7613 non-null   object
 4   target        7613 non-null   int64
 5   tweet_length  7613 non-null   int64
dtypes: int64(3), object(3)
memory usage: 357.0+ KB
None


DF Sample:

      target  \
216        0
7393       0
2298       0
492        1
723        0
386        1
6471       1
4794       0
4427       0
3067       0


                                                              text
216                                                             *to Luka* They
should all die! All of them! Everything annihilated! - Alois Trancy
7393         the windstorm blew thru my open window and now my bong is in pieces
just another example of nature's indifference to human suffering
2298  Just had my first counter on a league game against another Orianna I
happened to demolish her xD. I totally appreciate people that play her
492                     Christian Attacked by Muslims at the Temple Mount after
Waving Israeli Flag via Pamela Geller - … http://t.co/f5MiuhqaBy
723
@CoreyAshe Did that look broken or bleeding?
386                                   Mourning notices for stabbing arson
victims stir  Û÷politics of grief Ûª in Israel http://t.co/eug6zHciun
6471                   Japan FUSO Class Battleship YAMASHIRO Naval Cover 1999
PHOTO Cachet SUNK WWII http://t.co/Aq5ZliM7l4 http://t.co/FvR9jDQ71a
4794                 What the fuck was that. There was a loud bang and a flash
of light outside. I'm pretty sure I'm not dead but what the hell??
4427
```

```
I'm hungry as a hostage
3067
@devon_breneman hopefully it doesn't electrocute your heated blanket lmao


DF Location :

location
USA                    104
New York                71
United States           50
London                  45
Canada                  29
                      …
MontrÌ©al, QuÌ©bec       1
Montreal                 1
ÌÏT: 6.4682,3.18287      1
Live4Heed??              1
Lincoln                  1
Name: count, Length: 3341, dtype: int64


DF Keyword :

keyword
fatalities               45
deluge                   42
armageddon               42
sinking                  41
damage                   41
                         ..
forest%20fire            19
epicentre                12
threat                   11
inundation               10
radiation%20emergency     9
Name: count, Length: 221, dtype: int64
```

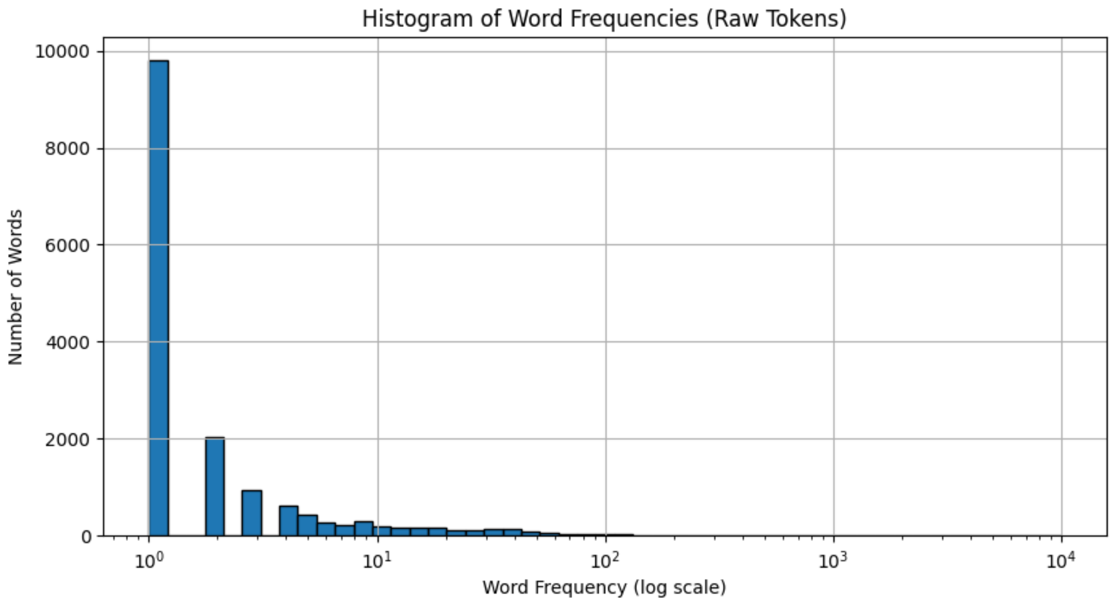
5.2 Word Frequency Analysis

A frequency analysis can provide insights into the distribution of words in the dataset. For example, are there a small number of words that are used most frequently (High peak, short tail), or does the distribution have a larger number of words that are used frequently (Low peak, long tail)?

To answer this question, I plotted a histogram of the word frequencies of the dataset. The y-axis represents the number of words that occur at a given frequency, while the x-axis represents the frequency of the words. The histogram shows that there are a large number of words that occur only once (about 10000), and a small number of words that occur more frequently. In this distribution, about 13,000 words of the 16,134 total words occur three times or less.

[13]:
```
# Word frequency analysis
def preprocessor(text):
    return re.findall(r'\b[a-z]{2,}\b', text.lower()) # Only keep words with 2␣
↪or more alphabetic characters

word_counter = Counter()
for text in df['text']:
    word_counter.update(preprocessor(text))

word_freqs = np.array(list(word_counter.values()))

# Count how many words occurred exactly once
singleton_count = sum(1 for count in word_counter.values() if count == 1)
print(f"Number of unique words that occur exactly once: {singleton_count}")

# Total unique words
total_unique_words = len(word_counter)
print(f"Total unique words in corpus: {total_unique_words}")
plt.figure(figsize=(10,5))
plt.hist(word_freqs, bins=np.logspace(0, 4, 50), edgecolor='black')
plt.xscale('log')
plt.xlabel('Word Frequency (log scale)')
plt.ylabel('Number of Words')
plt.title('Histogram of Word Frequencies (Raw Tokens)')
plt.grid(True)
plt.show()
```

```
Number of unique words that occur exactly once: 9792
Total unique words in corpus: 16134
```



5.3 Distribution of Tweet Lengths

The below histogram shows that the distribution is left-skewed, with almost all tweets being less than 29 words long. The mean tweet length is 14.5 tokens, and the median is 15.0 tokens. The tensorflow tokenizer will vectorize the data based on punctuation and spaces. Since the token count will vary based on the tweek, I will pad the sequences with zeros to a max length; thus, ensuring consistency of input shapes.

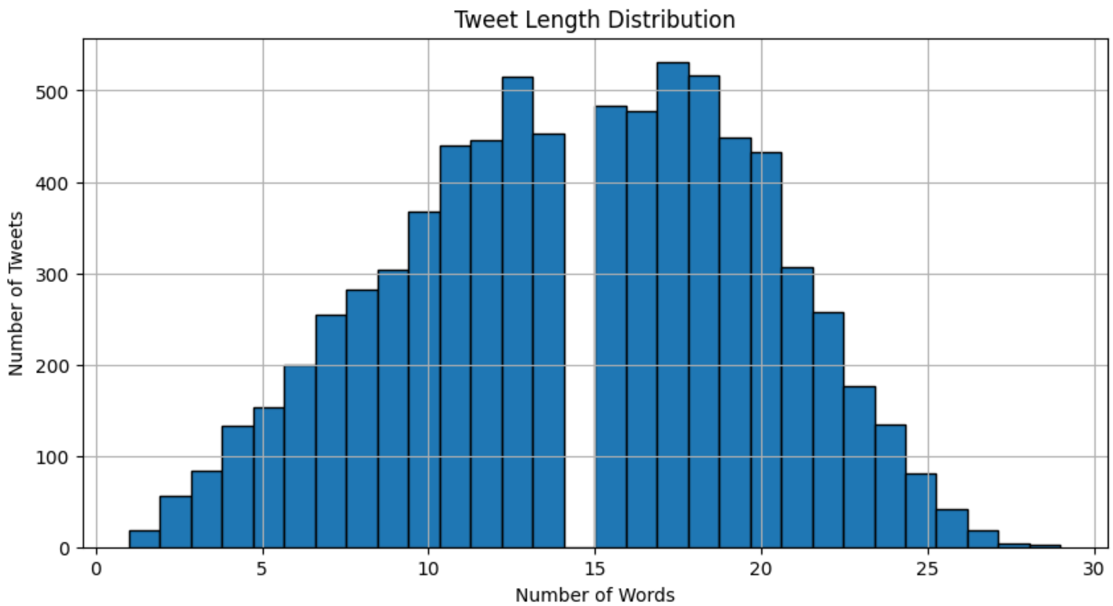
[12]:
```
df["text"].head()
```

[12]:
```
0                                                                 Our Deeds
are the Reason of this #earthquake May ALLAH Forgive us all
1
Forest fire near La Ronge Sask. Canada
2    All residents asked to 'shelter in place' are being notified by officers.
No other evacuation or shelter in place orders are expected
3                                                                     13,000
people receive #wildfires evacuation orders in California
4                                                 Just got sent this photo from
Ruby #Alaska as smoke from #wildfires pours into a school
Name: text, dtype: object
```

[14]:
```
df['tweet_length'] = df['text'].apply(lambda x: len(preprocessor(x)))
print(f"Average tweet length: {df['tweet_length'].mean()}")
print(f"Median tweet length: {df['tweet_length'].median()}")
print(f"Max tweet length: {df['tweet_length'].max()}")
print(f"Min tweet length: {df['tweet_length'].min()}")
```

```
plt.figure(figsize=(10,5))
plt.hist(df['tweet_length'], bins=30, edgecolor='black')
plt.xlabel('Number of Words')
plt.ylabel('Number of Tweets')
plt.title('Tweet Length Distribution')
plt.grid(True)
plt.show()
```

```
Average tweet length: 14.452778142650729
Median tweet length: 15.0
Max tweet length: 29
Min tweet length: 1
```



5.4 Sample Tweets

- **Low-Token Tweets** typically contain emojis, short phrases, or slang. I expect these tweets to be more challenging for the model to classify, as there isn't significant contextual information to extract.
- **High-Token Tweets** typically contain more complete sentences, more context, andd entire phrases/quotes. I expect these tweets to be easier for the model to classify, as there is more context and information to extract.

[17]:
```
## Samples of tweets with low word count
low_word_tweets = df[df['tweet_length'] < 5]
print("\n\nSamples of tweets with low word count:\n")
print(low_word_tweets[['text', 'target']].head(20))
```

```
Samples of tweets with low word count:

                                        text  target
15                              What's up man?       0
16                               I love fruits       0
17                             Summer is lovely       0
19                  What a goooooooaaaaaal!!!!!!       0
20                        this is ridiculous…       0
21                            London is cool ;)       0
22                                 Love skiing       0
23                         What a wonderful day!       0
24                                    LOOOOOOL       0
27                            Love my girlfriend       0
28                                  Cooool :)       0
29                            Do you like pasta?       0
30                                    The end!       0
39                        Ablaze for you Lord :D       0
73             BigRigRadio Live Accident Awareness       1
113             Aftershock https://t.co/xMWODFMtUI       0
130                  @OnFireAnders I love you bb       0
131             Aftershock https://t.co/jV8ppKhJY7       0
165                    I had a airplane accident.       1
214  Annihilated Abs . ?? http://t.co/1xPw292tJe       1
```

[19]:
```python
## Samples of tweets with high word count
high_word_tweets = df[df['tweet_length'] > 25]
print("\n\nSamples of tweets with high word count:\n")
print(high_word_tweets[['text', 'target']].head(10))
```

```
Samples of tweets with high word count:

                                                                      text  \
49         First night with retainers in. It's quite weird. Better get used to
it; I have to wear them every single night for the next year at least.
80   mom: 'we didn't get home as fast as we wished' \nme: 'why is that?'\nmom:
'there was an accident and some truck spilt mayonnaise all over ??????
734       Deadpool is already one of my favourite marvel characters and all I
know is he wears a red suit so the bad guys can't tell if he's bleeding
804       @parksboardfacts first off it is the #ZippoLine as no one wants to use
it and the community never asked for this blight on the park #moveit
806                  @anellatulip and put the taint there and that all that the
magisters did was to open the gates and let the blight get away from it
902  You know how they say the side effects low &amp; really fast? Son the
product was an acne cream.. Why 1 of the side effects was bloody diarrhea?
945        @PrincessDuck last week wanted the 6th sense to get blown up so far
```

```
so good. James could win but he's a huge target and will be gone soon.
954        If you have a son or a daughter would you like to see them going to a
war with Iran and come back in a body bag? Let the #Republicans know
965         @TR_jdavis Bruh you wanna fight I'm down meet me in the cage bro
better find out who you're dealing with before you end up in a body bag
990        Idgaf who tough or who from Canada and who from north Philly meek
been acting like a bitch &amp; drake been body bagging his ass on tracks

     target
49        0
80        0
734       0
804       0
806       0
902       0
945       0
954       0
965       0
990       0
```

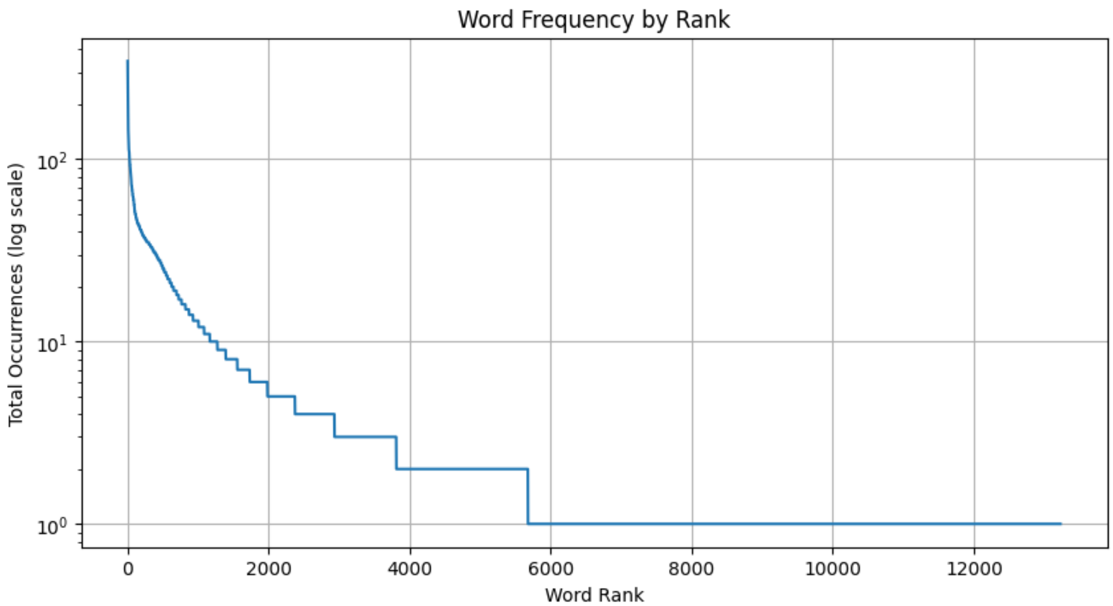
5.5 Word Frequency Distribution

To determine a starting point for the maximum number of features in the Tokenizer, I plotted the word frequency by rank. The x-axis represents the rank of the word (1 being the most frequent word, 2 being the second most frequent word, etc.). The y-axis represents the total number of times that word appears in the corpus.

I used the word frequency by rank to help inform my choice for max-features in the Tokenizer. The chart below shows that there is an elbow around 1000 words, with a drop off after about 5500 words. I will use the top 5677 words for the tokenizer as that's the number of words that appear more than once. I will use this as my first iteration of the tokenizer. This is a hyperparameter that can be tuned if the model's performance; however, increasing the number of features will increase the dimensionality of the data and may lead to overfitting.

[46]:
```python
# Word frequency distribution
vectorizer = CountVectorizer(stop_words='english')
X_counts = vectorizer.fit_transform(df['cleaned_text'])

word_freqs = np.asarray(X_counts.sum(axis=0)).ravel()
vocab = vectorizer.get_feature_names_out()

# Sort for display
sorted_freqs = np.sort(word_freqs)[::-1]

plt.figure(figsize=(10,5))
plt.plot(sorted_freqs)
plt.xlabel('Word Rank')
plt.ylabel('Total Occurrences (log scale)')
```

```
plt.title('Word Frequency by Rank')
plt.yscale('log')
plt.grid(True)
plt.show()

cutoff_idx = np.where(sorted_freqs == 1)[0][0]

print(f"First rank where frequency=1: {cutoff_idx+1}")  # +1 because rank is␣
↪1-based
print(f"Total vocab size: {len(sorted_freqs)}")
print(f"Words with frequency >1: {cutoff_idx}")
print(f"Words with frequency ==1: {len(sorted_freqs) - cutoff_idx}")
```



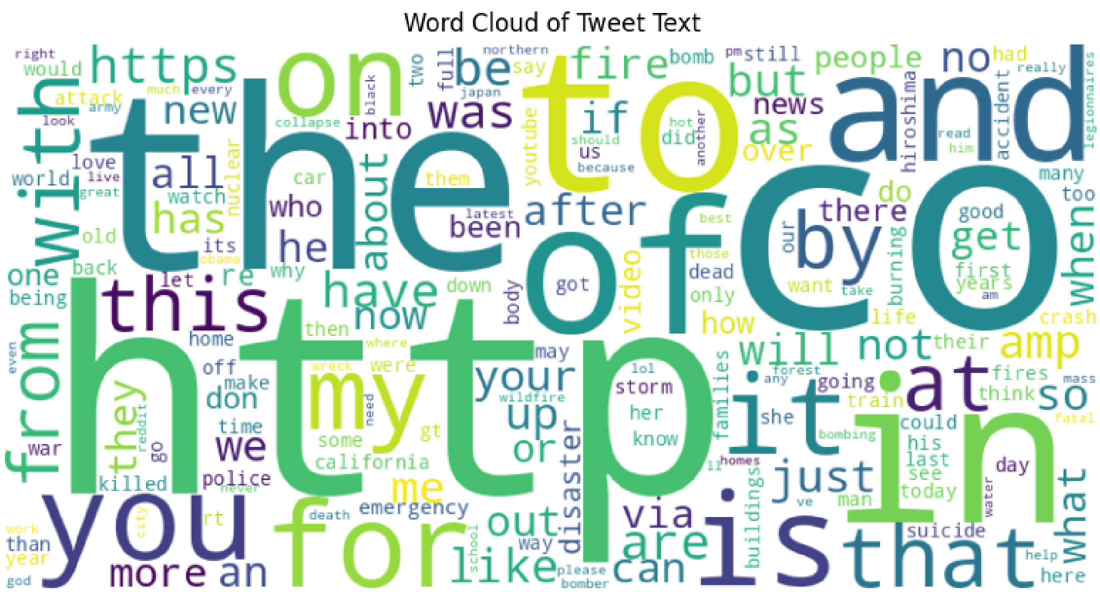

```
First rank where frequency=1: 5678
Total vocab size: 13230
Words with frequency >1: 5677
Words with frequency ==1: 7553
```

5.6 Word Cloud

To get a better understanding of the most common words before and after cleaning, I created a word cloud of the corpus. We can see that before cleaning, the word cloud contains many common words, such as “the”, “https”, “to”, etc. After cleaning, the word cloud contains more meaningful words that have the potential to provide the model with the necessary insights to classify the tweets.

[22]:

```
from wordcloud import WordCloud
wordcloud = WordCloud(width=800, height=400, background_color='white').
 ↪generate_from_frequencies(word_counter)
plt.figure(figsize=(10,5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Tweet Text')
plt.show()
```



5.7 Clean Text and Remove Stop words

Some of the tweets contain hyperlinks, which are not useful for our analysis. These will be removed in the preprocessing step before the TFIDF vectorizor or mebeding models are applied to the corpus. Additionally, english stop words will be removed to reduce uninformative words in the dataset.

- ☒ Remove hyperlinks
- ☒ Remove stop words
- ☒ Remove punctuation
- ☒ Remove numbers
- ☒ Remove special characters

```
[ ]: def clean_tweet(text):
         # Remove emojis (unicode ranges covering emoticons, symbols, pictographs,␣
      ↪flags, etc.)
         emoji_pattern = re.compile(
             "["
             "\U0001F600-\U0001F64F"  # emoticons
```

```
        "\U0001F300-\U0001F5FF"  # symbols & pictographs
        "\U0001F680-\U0001F6FF"  # transport & map symbols
        "\U0001F1E0-\U0001F1FF"  # flags
        "\U00002500-\U00002BEF"  # chinese characters
        "\U00002702-\U000027B0"
        "\U000024C2-\U0001F251"
        "\U0001f926-\U0001f937"
        "\U00010000-\U0010ffff"
        "\u200d"
        "\u2640-\u2642"
        "\u2600-\u2B55"
        "\u23cf"
        "\u23e9"
        "\u231a"
        "\ufe0f"
        "\u3030"
        "]+",
        flags=re.UNICODE
    )

    text = emoji_pattern.sub(r'', text)                 # remove emojis
    text = re.sub(r'\d+', '', text)                        # remove numbers
    text = re.sub(r'http\S+', '', text)                 # remove URLs
    text = re.sub(r'@\w+', '', text)                    # remove mentions
    text = re.sub(r'#\w+', '', text)                    # remove hashtags
    text = re.sub(r'[^A-Za-z0-9\s]+', '', text)         # remove punctuation/
↪special chars
    text = text.lower().strip()                         # lowercase and strip
    return text

df['cleaned_text'] = df['text'].apply(clean_tweet)
df_test['cleaned_text'] = df_test['text'].apply(clean_tweet)
```

[41]:
```
print(df["text"].sample(20))
```

```
3613        11-Year-Old Boy Charged With Manslaughter of Toddler: Report: An
11-year-old boy has been charged with manslaughter over the fatal sh…
1816        Bin Laden family plane crashed after 'avoiding microlight and
landing too far down runway': Three members of t… http://t.co/mFJxh4p51U
3716                          I want to be with you forever\nStay by my side\nOn
this special night\n\nFear and Loathing in Las Vegas/Solitude X'mas
5012                              STERLING-SCOTT on the Red Carpet at a
fundraiser for 'OSO Mudslide' https://t.co/mA4ra7AtqL http://t.co/cg579wlDnE
3134                                                        Emergency Flow
http://t.co/lH9mrYpDrJ mp3 http://t.co/PquthSS3i rar http://t.co/0iW6dRf5X9
7346                                                California is battling its
scariest 2015 wildfire so far - the Rocky Fire http://t.co/sPT54KfA9Q
```

```
3497                              #Tampa: Super Freestyle Explosion Live in
Concert at Amalie Arena - Sep  19\n? Ticket Info: http://t.co/ooGotO76uZ
7106       U.S.PACIFIC COMMAND.\nI can see it!\nThey gave their all in the peace
unity festival\nIt disappears when freedom\nA Violent Storm hit Sea
5755                                                     I liked a @YouTube video
http://t.co/5fR41TPzte Thorin's Thoughts - Riot and Sandbox Mode (LoL)
6222                    If you wanna smoke cigs that's your own problem but when
your breath smells like an old ash tray.. that's fucking disgusting
6579                 Molecularly targeted cancer therapy for his #LungCancer
gave Rocky his life back. http://t.co/TwI3pYm7Us http://t.co/qT8JMD9pI1
3593       California man facing manslaughter charge in Sunday's wrong-way fatal
crash in … - http://t.co/1vz3RmjHy4: Ca… http://t.co/xevUEEfQBZ
2793                               Blue Bell May Be Close to a Return From Its
Listeria Disaster… Hot on #theneeds #Recipes http://t.co/F56v61AmPt
607     USATODAY: On today's #frontpage: #Bioterror lab faced secret sanctions.
#RickPerry doesn't make the cut for FoxNew Û_ http://t.co/xFHh2XF9Ga
6837        Hollywood Movie About Trapped Miners Released in Chile: 'The 33'
Hollywood movie about trapped miners starring… http://t.co/tyyfG4qQvM
6830        @dramaa_llama but otherwise i will stay trapped as the worst
lilourry stan ever AND without zarry what am I left with? NARRY. NO THANKS.
7333              Drones Under Fire: Officials Offer $75000 Reward Leading To
Pilots Who Flew Over Wildfire http://t.co/d2vEppeh8S #photography #arts
5364                @JetixRestored Here's Part 2 Of Teamo Supremo Pogo Panic! I
Want You Make It Better! OK! :) https://t.co/wBLiMlMT2x via @YouTube
5136                                                 Finnish ministers: Fennovoima
nuclear reactor will go ahead via /r/worldnews http://t.co/fRkOdEstuK
7032                                                         Obama Declares Disaster
for Typhoon-Devastated Saipan http://t.co/CanEyTtwEV #international
Name: text, dtype: object
```

[42]:
```python
# Verify hyperlinks removed
id = 6626
print(f"Original tweet: {df.iloc[id]['text']}")
print(f"Cleaned tweet: {df.iloc[id]['cleaned_text']}")

# Verify only english characters
id = 3710
print(f"Original tweet: {df.iloc[id]['text']}")
print(f"Cleaned tweet: {df.iloc[id]['cleaned_text']}")

# Verify numbers removed
id = 7548
print(f"Original tweet: {df.iloc[id]['text']}")
print(f"Cleaned tweet: {df.iloc[id]['cleaned_text']}")
```

```
Original tweet: Truth…
https://t.co/4ZQrsAQrRT
#News
```

```
#BBC
#CNN
#Islam
#Truth
#god
#ISIS
#terrorism
#Quran
#Lies http://t.co/6ar3UKvsxw
Cleaned tweet: truth
Original tweet: @BaileySMSteach The fear of not looking like you know what you
are doing Û_ahhh Û_that was a big one for me.  #PerryChat
Cleaned tweet: the fear of not looking like you know what you are doingahhhthat
was a big one for me
Original tweet: Four hundred wrecked cars (costing $100 apiece) were purchased
for the making of this 1986 film - http://t.co/DTdidinQyF
Cleaned tweet: four hundred wrecked cars costing  apiece were purchased for the
making of this  film
```

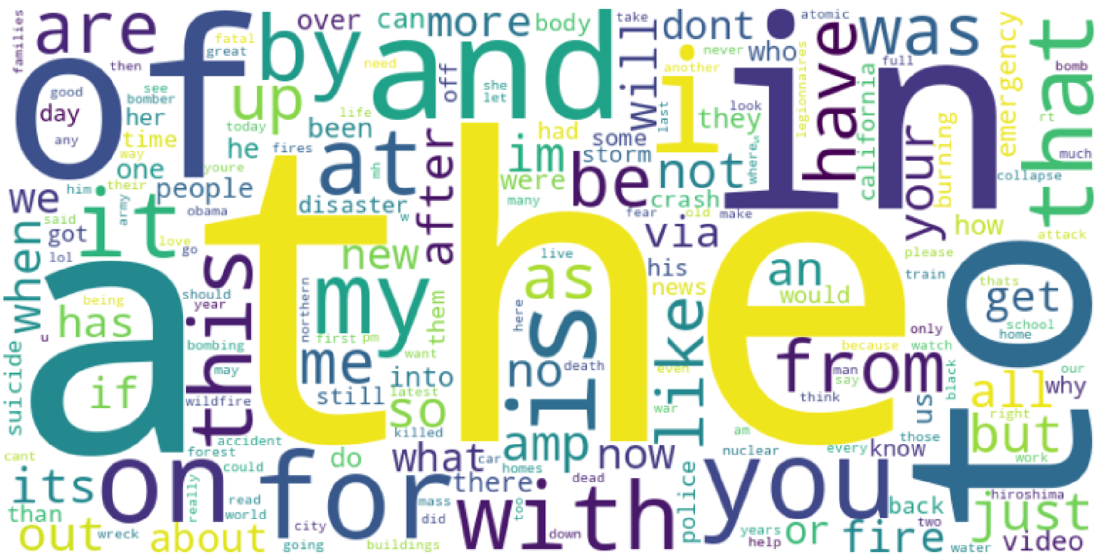
5.8 Word Cloud after cleaning

After removing hyperlinks, stop words, punctuation, numbers, special characters, and emojis. I counted the words with Counter.

As expected, there are mostly 'stop' words, like 'the', 'and', 'of', etc… Since I am using Neural Networks, I am hypothesising that these words staying in the tweets shouldn't reduce the performance as the models can learn to ignore them since the models inherently learn language patterns.

[43]:
```
word_counter_cleaned = Counter()
for text in df['cleaned_text']:
    word_counter_cleaned.update(text.split())

wordcloud_cleaned = WordCloud(width=800, height=400, background_color='white').
 ↪generate_from_frequencies(dict(word_counter_cleaned))

plt.figure(figsize=(10,5))
plt.imshow(wordcloud_cleaned, interpolation='bilinear')
plt.axis('off')
plt.title('Word Cloud of Tweet Text')
plt.show()
```

Word Cloud of Tweet Text



6 Feature Engineering

6.1 Train-validation split

In the below cell. I'll split the data into a training and validation set. The training and validation set will be used to train and validate the epochs in the model. The split will be 80% for training and 20% for validation. Despite being balanced, I'll still use stratified sampling to ensure that both sets have the same distribution of labels as the original dataset.

[45]:
```
# Train-validation split

df_train, df_val = train_test_split(df, test_size=0.2, random_state=42,␣
 ↪stratify=df['target'])
print("\n\nTrain set shape:", df_train.shape)
print("Validation set shape:", df_val.shape)
print("Train set target distribution:\n", df_train['target'].
 ↪value_counts(normalize=True))
print("Train set target distribution:\n", df_train['target'].mean())
print("Validation set target distribution:\n", df_val['target'].
 ↪value_counts(normalize=True))
print("Validation set target distribution:\n", df_val['target'].mean())
```

```
Train set shape: (6090, 7)
Validation set shape: (1523, 7)
Train set target distribution:
```

```
 target
0    0.570279
1    0.429721
Name: proportion, dtype: float64
Train set target distribution:
 0.4297208538587849
Validation set target distribution:
 target
0    0.570584
1    0.429416
Name: proportion, dtype: float64
Validation set target distribution:
 0.4294156270518713
```

6.2 Vectorization

I will use tensorflow.keras.preprocessing.text.Tokenizer for text tokenization. This splits the cleaned text values on puncuation and whitespace. After tokenizing the values, I will then pad the sequences to ensure uniform input size for the neural network. On the padded values, I will use a pretrained embedding model (word2vec) to provide the final features for the LSTM/GRU model with a binary classifier head. After using that initial embedding model, I will attempt to fine tune the embedding layer. I will also use a more specialized embedding model, GLoVe, which was trained on twitter data. I'm hoping that by adjusting the emnbedding layer, along with the RNN architecture, I can create a model that will be able to classify on tweets, which are typically short and contain slang, emojis, hyperlinks, and tag.

[ ]:
```
## Tokenize the cleaned text from the tweet dataset
tokenizer = Tokenizer(num_words=5677)  # 5677 is from the frequency␣
 ↪distribution chart
tokenizer.fit_on_texts(df_train["cleaned_text"])

train_sequences = tokenizer.texts_to_sequences(df_train["cleaned_text"])
train_padded = pad_sequences(train_sequences, padding="post", maxlen=100)

val_sequences = tokenizer.texts_to_sequences(df_val["cleaned_text"])
val_padded = pad_sequences(val_sequences, padding="post", maxlen=100)

sample_word_index = random.sample(list(tokenizer.word_index.items()), 10)
print(dict(sample_word_index))  # print first 10 words
```

```
{'forever': 1892, 'trillion': 3987, 'remove': 1360, 'immigration': 11457,
'dominance': 8289, 'sedar': 6247, 'moores': 10758, 'lives': 293, 'reddit': 300,
'active': 2660}
```

6.3 Embedding

I will initially use the pretrained Word2Vec[2] embedding model. This model was trained with skip-grams and has a dimensionality of 300. I chose the embedding model because there was a lot of infrequently used terms/slang and I wanted to be able to capture their uniqueness in context, not the conventionally agreed upon definition of words in more structured context. In the hyper parameter tuning section, I experiment with GLoVe embeddings and fine-tuning the embedding layer. However, I think that this embedding model will be a good starting point for a baseline model and architectural decisions.

[60]:
```
import gensim.downloader as api

# if embedding_matrix doesn't exist
if not os.path.exists(f"{DATA_DIR}/embedding_matrix.npy"):
    w2v_model = api.load("word2vec-google-news-300")

    ## Map the word index to the embedding matrix
    embedding_dim = 300
    embedding_matrix = np.zeros((len(tokenizer.word_index) + 1, embedding_dim))

    for word, i in tokenizer.word_index.items():
        embedding_vector = w2v_model[word] if word in w2v_model else None
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
    # Save the embedding matrix
    np.save(f"{DATA_DIR}/embedding_matrix.npy", embedding_matrix)
else:
    embedding_matrix = np.load(f"{DATA_DIR}/embedding_matrix.npy")

# Verify embedding matrix
print("Expected shape of embedding matrix:", (len(tokenizer.word_index) + 1,␣
 ↪300))
print("Embedding matrix shape:", embedding_matrix.shape)
```

```
Expected shape of embedding matrix: (11970, 300)
Embedding matrix shape: (11970, 300)
```

7 LSTM Model

The first architecture that I will use is a simple LSTM model. These networks are designed to handle sequential data and can capture long-term information in a way that traditional RNNs can't. To use this model, I tokenized the input data and padded the sequences to ensure consistent input shapes. This will be the baseline model that I will compare a more complex LSTM, GRU, fine-tuned embedding layer, and different embedding models against. I will compare the impact of the changes in the results section along with discussion of future work.

The LSTM layers contain three gates: - The **input gate** determines what new information to add to the cell state. - The **forget gate** determines what information to disregard from the previous cell state. - The **output gate** determines what information to output from the current cell state.

I chose a single LSTM layer with 64 units. Since the padded sequence was 100, I thought that 64 units would be sufficent to capture the information in the tweets without overfitting. I then added a dropout layer to reduce the chance of overfitting. The final layer is a dense layer with 16 neurons followed by a sigmoid activation function to output a binary classification, as the classification task is binary (disaster or non-disaster).

The results of this model are: |Model Name| Training Accuracy | Training Precision | Training Recall | Validation Accuracy | Kaggle Test Accuracy | |—|—|-|-|—|—| |LSTM Baseline|0.83|0.85|0.82|0.81| 0.79619|

[75]:
```
from tensorflow.keras import layers, models, optimizers, callbacks

# import sequential
from keras.models import Sequential

vocab_size = len(tokenizer.word_index) + 1
assert (
    vocab_size == embedding_matrix.shape[0]
), "Vocab size and embedding matrix size do not match"

seq_len = train_padded.shape[1]  # 100
assert (
    train_padded.shape[1] == val_padded.shape[1]
), "Training and validation data sequence lengths do not match"

input_layer = layers.Input(shape=(seq_len,), dtype="int32")

embedding_layer_frozen = layers.Embedding(
    input_dim=vocab_size,
    output_dim=300,
    weights=[embedding_matrix],
    input_length=seq_len,
    mask_zero=True,  # Prevents RNN from updating on padded values
    trainable=False,  # Potential to unfreeze this later.
)

lstm_layer = layers.Bidirectional(layers.LSTM(64, return_sequences=False))

dropout_layer = layers.Dropout(0.3)

dense_layer = layers.Dense(16, activation="relu")


output_layer = layers.Dense(1, activation="sigmoid")

model = models.Sequential(
    [
        input_layer,
        embedding_layer_frozen,
        lstm_layer,
        dropout_layer,
```

```
        dense_layer,
        output_layer,
    ]
)

model.compile(
    optimizer=optimizers.Adam(learning_rate=1e-4),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)

model.summary()

early_stopping = callbacks.EarlyStopping(
    monitor="val_loss",
    patience=5,
    restore_best_weights=True,
)
```

```
Model: "sequential_3"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 100, 300)          3591000

 bidirectional_3 (Bidirecti  (None, 128)               186880
 onal)

 dropout (Dropout)           (None, 128)               0

 dense_6 (Dense)             (None, 16)                2064

 dense_7 (Dense)             (None, 1)                 17

=================================================================
Total params: 3779961 (14.42 MB)
Trainable params: 188961 (738.13 KB)
Non-trainable params: 3591000 (13.70 MB)
_________________________________________________________________
```

[ ]:
```
model_name = "LSTM_Frozen_Embeddings"
if os.path.exists(f"{DATA_DIR}/models/{model_name}.h5"):
    model, history = load_model_and_history(model_name)
else:
    history = model.fit(
        train_padded,
        df_train["target"].values,
```

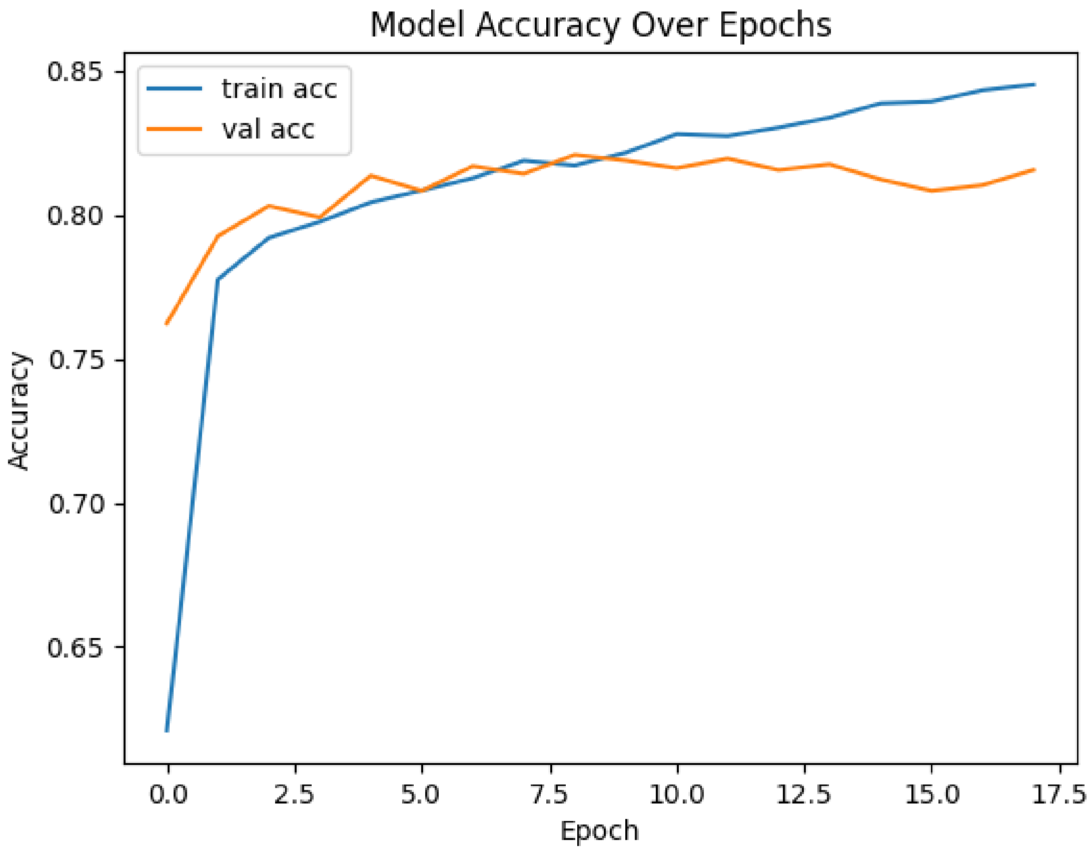
```
        epochs=30,
        batch_size=32,
        validation_data=(val_padded, df_val["target"].values),
        callbacks=[early_stopping],
)
```

```
Epoch 1/30
191/191 [==============================] - 16s 58ms/step - loss: 0.6376 -
accuracy: 0.6210 - val_loss: 0.5816 - val_accuracy: 0.7623
Epoch 2/30
191/191 [==============================] - 11s 50ms/step - loss: 0.5304 -
accuracy: 0.7775 - val_loss: 0.4821 - val_accuracy: 0.7925
Epoch 3/30
191/191 [==============================] - 10s 54ms/step - loss: 0.4724 -
accuracy: 0.7920 - val_loss: 0.4594 - val_accuracy: 0.8030
Epoch 4/30
191/191 [==============================] - 10s 55ms/step - loss: 0.4521 -
accuracy: 0.7975 - val_loss: 0.4570 - val_accuracy: 0.7991
Epoch 5/30
191/191 [==============================] - 11s 56ms/step - loss: 0.4377 -
accuracy: 0.8043 - val_loss: 0.4441 - val_accuracy: 0.8135
Epoch 6/30
191/191 [==============================] - 12s 62ms/step - loss: 0.4305 -
accuracy: 0.8084 - val_loss: 0.4476 - val_accuracy: 0.8083
Epoch 7/30
191/191 [==============================] - 12s 61ms/step - loss: 0.4215 -
accuracy: 0.8126 - val_loss: 0.4356 - val_accuracy: 0.8168
Epoch 8/30
191/191 [==============================] - 12s 64ms/step - loss: 0.4153 -
accuracy: 0.8187 - val_loss: 0.4454 - val_accuracy: 0.8142
Epoch 9/30
191/191 [==============================] - 10s 51ms/step - loss: 0.4111 -
accuracy: 0.8171 - val_loss: 0.4329 - val_accuracy: 0.8207
Epoch 10/30
191/191 [==============================] - 11s 56ms/step - loss: 0.4086 -
accuracy: 0.8215 - val_loss: 0.4310 - val_accuracy: 0.8188
Epoch 11/30
191/191 [==============================] - 11s 56ms/step - loss: 0.3983 -
accuracy: 0.8279 - val_loss: 0.4332 - val_accuracy: 0.8162
Epoch 12/30
191/191 [==============================] - 11s 58ms/step - loss: 0.3961 -
accuracy: 0.8273 - val_loss: 0.4412 - val_accuracy: 0.8194
Epoch 13/30
191/191 [==============================] - 11s 56ms/step - loss: 0.3901 -
accuracy: 0.8302 - val_loss: 0.4309 - val_accuracy: 0.8155
Epoch 14/30
191/191 [==============================] - 11s 55ms/step - loss: 0.3830 -
```

```
accuracy: 0.8337 - val_loss: 0.4356 - val_accuracy: 0.8175
Epoch 15/30
191/191 [==============================] - 11s 58ms/step - loss: 0.3767 -
accuracy: 0.8386 - val_loss: 0.4550 - val_accuracy: 0.8122
Epoch 16/30
191/191 [==============================] - 11s 60ms/step - loss: 0.3759 -
accuracy: 0.8392 - val_loss: 0.4463 - val_accuracy: 0.8083
Epoch 17/30
191/191 [==============================] - 11s 53ms/step - loss: 0.3711 -
accuracy: 0.8432 - val_loss: 0.4463 - val_accuracy: 0.8102
Epoch 18/30
191/191 [==============================] - 10s 54ms/step - loss: 0.3650 -
accuracy: 0.8452 - val_loss: 0.4434 - val_accuracy: 0.8155
```

[77]:

```
# Plot epoch history

plt.plot(history.history["accuracy"], label="train acc")
plt.plot(history.history["val_accuracy"], label="val acc")
plt.title("Model Accuracy Over Epochs")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



[ ]:
```python
# Save the model
from io import StringIO
model_name = "LSTM_Frozen_Embeddings"
print(f"Model Name: {model_name}")
model.save(f"{MODEL_DIR}/{model_name}.h5")

import pickle

with open(f"{MODEL_DIR}/{model_name}_history.pkl", "wb") as f:
    pickle.dump(history.history, f)

# Capture model summary
summary_buffer = StringIO()
model.summary(print_fn=lambda x: summary_buffer.write(x + '\n'))
model_summary_str = summary_buffer.getvalue()
summary_buffer.close()

# Write to file
with open(f"{MODEL_DIR}{model_name}_performance.txt", "w") as f:
    f.write(f"Model Name: {model_name}\n")
```

```
    f.write(f"Input Shape: {model.input_shape[1:]}\n")
    f.write(f"Total Parameters: {model.count_params() // 1000}K\n")
    f.write(f"Optimizer: Adam\n")
    f.write(f"Training Accuracy: {history.history['accuracy'][-1]:.4f}\n")
    f.write(f"Validation Accuracy: {history.history['val_accuracy'][-1]:.4f}\n")
    f.write("Model Summary:\n")
    f.write(model_summary_str)
```

```
Model Name: LSTM_Frozen_Embeddings
```

```
/home/megarnol/projects/MSDS_Notes_Playground/.venv/lib/python3.10/site-
packages/keras/src/engine/training.py:3103: UserWarning: You are saving your
model as an HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

7.1 Results for LSTM Baseline

[166]:
```
# Predictions of test dataset
test_predictions, submissions = get_test_predictions(
    model=model,
    tokenizer=tokenizer,
    max_length=100,
    texts=df_test["cleaned_text"].tolist(),
    ids=df_test["id"].tolist(),
)
model_name = "LSTM_Frozen_Embeddings"
test_predictions.to_csv(f"{DATA_DIR}/submissions/{model_name}_test_predictions.
 ↪csv", index=False)
submissions.to_csv(f"{DATA_DIR}/submissions/{model_name}_submission.csv",␣
 ↪index=False)
```

```
102/102 [==============================] - 2s 19ms/step
```

[174]:
```
# Get train predictions
train_predictions, _ = get_test_predictions(
    model=model,
    tokenizer=tokenizer,
    max_length=100,
    texts=df_train["cleaned_text"].tolist(),
    ids=df_train["id"].tolist(),
)
model_name = "LSTM_Frozen_Embeddings"
train_predictions["actual"] = df_train["target"].values
train_predictions.to_csv(f"{DATA_DIR}/submissions/
 ↪{model_name}_train_predictions.csv", index=False)
```

```python
tp_samples, fp_samples, tn_samples, fn_samples =␣
 ↪get_tp_fp_tn_fn_samples(train_predictions)

print(f"Classification Report:\n")
print(classification_report(train_predictions["actual"],␣
 ↪train_predictions["target"]))
print(f"True Positive Tweet Samples: \n{tp_samples}\n")
print(f"False Positive Tweet Samples: \n{fp_samples}\n")
print(f"True Negative Tweet Samples: \n{tn_samples}\n")
print(f"False Negative Tweet Samples: \n{fn_samples}\n")
```

```
191/191 [==============================] - 2s 12ms/step
Classification Report:

              precision    recall  f1-score   support

           0       0.81      0.93      0.87      3473
           1       0.89      0.70      0.78      2617

    accuracy                           0.83      6090
   macro avg       0.85      0.82      0.82      6090
weighted avg       0.84      0.83      0.83      6090

True Positive Tweet Samples:
4308           killed in sarabia mosque suicide bombing\n\na suicide bomber
attacked a mosque in aseer southwestern saudi
3110        former township fire truck being used in philippines township of
langley assistant fire chief pat walker spen
2690    investigators say a fatal virgin galactic spaceship crash last year was
caused by structural failure after the co
Name: text, dtype: object

False Positive Tweet Samples:
3167                                                          the lightning
strike de snow patrol de a hundred million suns
2200
fatality
2000    all obama is doing is giving a false time schedule on iran testing there
first bomb      bomb  nuclear suicide vest
Name: text, dtype: object

True Negative Tweet Samples:
3168                           the trouble in one of buffetts favorite sectors
3859                 fatal attraction is common n what we have common is pain
4218    suicide of a superpower  will america survive to  by patrick j buchana
Name: text, dtype: object
```

```
False Negative Tweet Samples:
3382                               saw that pileup on tv keep racing even
bleeding
2166    the gusto in persist had amongst emptypated communication explosion
hpssjd
5667                     call for tasmanias emergency services to be trained in
horse
Name: text, dtype: object
```

8 LSTM with Additional Regularization

Since the baseline model had some overfitting, that began around epoch 8, I chose to do additional regularization to see if it would improve the model's ability to generalize. I added recurrent dropout and input dropout to the LSTM layer. This was added by including the parameters `recurrent_dropout=0.2` and `dropout=0.2` to the LSTM layer. This will randomly drop 20% of the input and recurrent connections during training, which can help prevent overfitting.

Once this model was trained to 20 epochs, I saw that the model didn't overfit as much as the baseline model. The validation accuracy was actually higher than the training accuracy at the start of the training. I expect this model to perform similarly to the baseline model on the Kaggle test set.

The results of this model are: |Model Name| Training Accuracy | Training Precision | Training Recall | Validation Accuracy | Kaggle Test Accuracy | |—|—|-|-|—|—| |LSTM Baseline|0.83|0.85|0.82|0.81| 0.79619| |LSTM with Additional Dropout|0.84|0.83|0.83|0.81| 0.79129|

[80]:
```
from tensorflow.keras import layers, models, optimizers, callbacks

# import sequential
from keras.models import Sequential

vocab_size = len(tokenizer.word_index) + 1
assert (
    vocab_size == embedding_matrix.shape[0]
), "Vocab size and embedding matrix size do not match"

seq_len = train_padded.shape[1]  # 100
assert (
    train_padded.shape[1] == val_padded.shape[1]
), "Training and validation data sequence lengths do not match"

input_layer = layers.Input(shape=(seq_len,), dtype="int32")

embedding_layer_frozen = layers.Embedding(
    input_dim=vocab_size,
    output_dim=300,
    weights=[embedding_matrix],
```

```python
    input_length=seq_len,
    mask_zero=True,  # Prevents RNN from updating on padded values
    trainable=False,  # Potential to unfreeze this later.
)

lstm_layer_dropout = layers.Bidirectional(layers.LSTM(64, dropout = 0.2,␣
↪recurrent_dropout=0.2, return_sequences=False))

dropout_layer = layers.Dropout(0.3)

dense_layer = layers.Dense(16, activation="relu")


output_layer = layers.Dense(1, activation="sigmoid")

model_lstm_dropout = models.Sequential(
    [
        input_layer,
        embedding_layer_frozen,
        lstm_layer_dropout,
        dropout_layer,
        dense_layer,
        output_layer,
    ]
)

model_lstm_dropout.compile(
    optimizer=optimizers.Adam(learning_rate=1e-4),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)

model_lstm_dropout.summary()

early_stopping = callbacks.EarlyStopping(
    monitor="val_loss",
    patience=5,
    restore_best_weights=True,
)
```

```
WARNING:tensorflow:Layer lstm_4 will not use cuDNN kernels since it doesn't meet
the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer lstm_4 will not use cuDNN kernels since it doesn't meet
the criteria. It will use a generic GPU kernel as fallback when running on GPU.
WARNING:tensorflow:Layer lstm_4 will not use cuDNN kernels since it doesn't meet
the criteria. It will use a generic GPU kernel as fallback when running on GPU.
Model: "sequential_4"
```

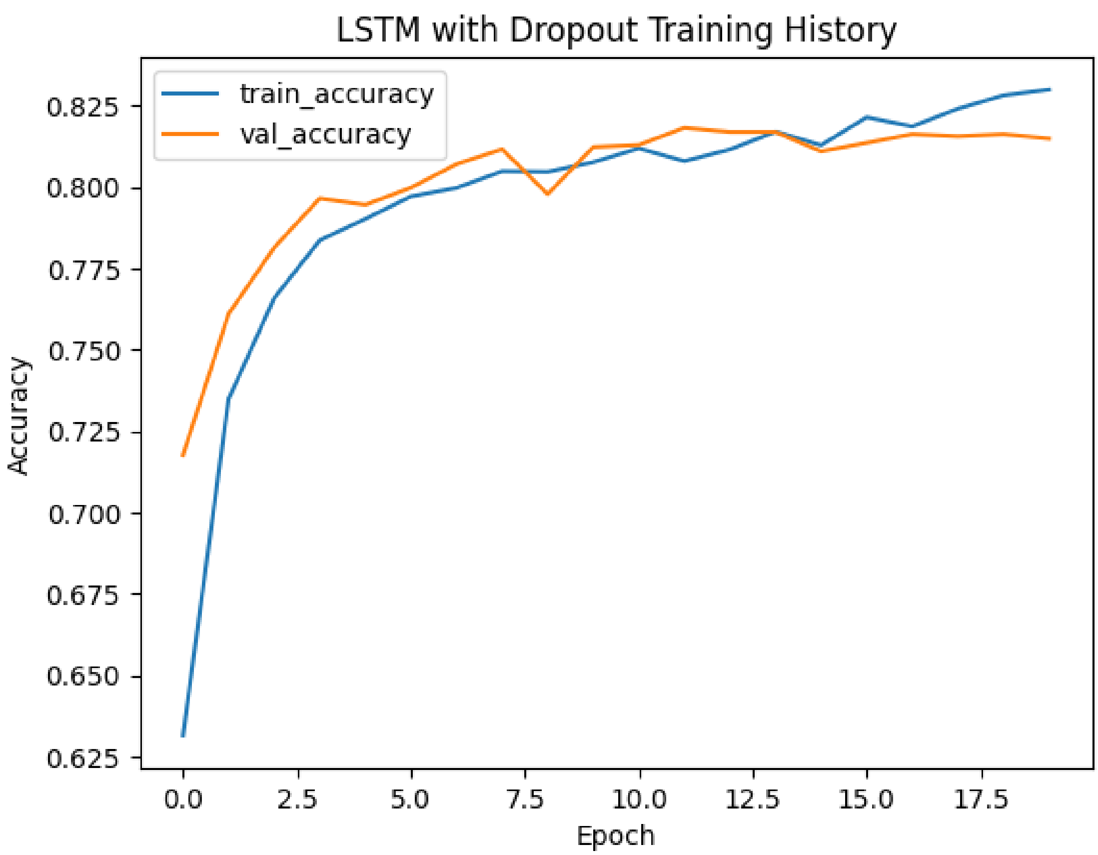
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_4 (Embedding)     (None, 100, 300)          3591000

 bidirectional_4 (Bidirecti  (None, 128)               186880
 onal)

 dropout_1 (Dropout)         (None, 128)               0

 dense_8 (Dense)             (None, 16)                2064

 dense_9 (Dense)             (None, 1)                 17

=================================================================
Total params: 3779961 (14.42 MB)
Trainable params: 188961 (738.13 KB)
Non-trainable params: 3591000 (13.70 MB)
_________________________________________________________________
```

[ ]:
```python
model_name = "LSTM_Frozen_Embeddings_Additional_Dropout"
if os.path.exists(f"{DATA_DIR}/models/{model_name}.h5"):
    model, history = load_model_and_history(model_name)
else:
    history = model.fit(
        train_padded,
        df_train["target"].values,
        validation_data=(val_padded, df_val["target"].values),
        epochs=20,
    batch_size=64,
    callbacks=[early_stopping],
)
```

```
Epoch 1/20
96/96 [==============================] - 160s 2s/step - loss: 0.6541 - accuracy:
0.6315 - val_loss: 0.6038 - val_accuracy: 0.7177
Epoch 2/20
96/96 [==============================] - 147s 2s/step - loss: 0.5689 - accuracy:
0.7348 - val_loss: 0.5160 - val_accuracy: 0.7610
Epoch 3/20
96/96 [==============================] - 145s 2s/step - loss: 0.5014 - accuracy:
0.7658 - val_loss: 0.4770 - val_accuracy: 0.7814
Epoch 4/20
96/96 [==============================] - 144s 2s/step - loss: 0.4729 - accuracy:
0.7836 - val_loss: 0.4640 - val_accuracy: 0.7965
Epoch 5/20
96/96 [==============================] - 144s 2s/step - loss: 0.4591 - accuracy:
0.7901 - val_loss: 0.4549 - val_accuracy: 0.7945
```

```
Epoch 6/20
96/96 [==============================] - 137s 1s/step - loss: 0.4496 - accuracy:
0.7970 - val_loss: 0.4565 - val_accuracy: 0.7997
Epoch 7/20
96/96 [==============================] - 129s 1s/step - loss: 0.4471 - accuracy:
0.7997 - val_loss: 0.4508 - val_accuracy: 0.8070
Epoch 8/20
96/96 [==============================] - 127s 1s/step - loss: 0.4368 - accuracy:
0.8048 - val_loss: 0.4453 - val_accuracy: 0.8116
Epoch 9/20
96/96 [==============================] - 129s 1s/step - loss: 0.4338 - accuracy:
0.8046 - val_loss: 0.4578 - val_accuracy: 0.7978
Epoch 10/20
96/96 [==============================] - 127s 1s/step - loss: 0.4296 - accuracy:
0.8076 - val_loss: 0.4489 - val_accuracy: 0.8122
Epoch 11/20
96/96 [==============================] - 125s 1s/step - loss: 0.4249 - accuracy:
0.8118 - val_loss: 0.4482 - val_accuracy: 0.8129
Epoch 12/20
96/96 [==============================] - 125s 1s/step - loss: 0.4242 - accuracy:
0.8079 - val_loss: 0.4365 - val_accuracy: 0.8181
Epoch 13/20
96/96 [==============================] - 127s 1s/step - loss: 0.4222 - accuracy:
0.8115 - val_loss: 0.4367 - val_accuracy: 0.8168
Epoch 14/20
96/96 [==============================] - 126s 1s/step - loss: 0.4142 - accuracy:
0.8169 - val_loss: 0.4355 - val_accuracy: 0.8168
Epoch 15/20
96/96 [==============================] - 127s 1s/step - loss: 0.4173 - accuracy:
0.8128 - val_loss: 0.4376 - val_accuracy: 0.8109
Epoch 16/20
96/96 [==============================] - 126s 1s/step - loss: 0.4113 - accuracy:
0.8213 - val_loss: 0.4361 - val_accuracy: 0.8135
Epoch 17/20
96/96 [==============================] - 129s 1s/step - loss: 0.4074 - accuracy:
0.8186 - val_loss: 0.4324 - val_accuracy: 0.8162
Epoch 18/20
96/96 [==============================] - 128s 1s/step - loss: 0.4054 - accuracy:
0.8240 - val_loss: 0.4305 - val_accuracy: 0.8155
Epoch 19/20
96/96 [==============================] - 128s 1s/step - loss: 0.4004 - accuracy:
0.8281 - val_loss: 0.4343 - val_accuracy: 0.8162
Epoch 20/20
96/96 [==============================] - 127s 1s/step - loss: 0.3947 - accuracy:
0.8299 - val_loss: 0.4428 - val_accuracy: 0.8148
```

[82]:
```
# plot training history
plt.plot(history.history["accuracy"], label="train_accuracy")
plt.plot(history.history["val_accuracy"], label="val_accuracy")
plt.title("LSTM with Dropout Training History")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



[ ]:
```
# # Save the model
from io import StringIO
model_name = "LSTM_Frozen_Embeddings_Additional_Dropout"
print(f"Model Name: {model_name}")
model_lstm_dropout.save(f"{MODEL_DIR}/{model_name}.h5")

import pickle

with open(f"{MODEL_DIR}/{model_name}_history.pkl", "wb") as f:
    pickle.dump(history.history, f)
```

```python
# Capture model summary
summary_buffer = StringIO()
model_lstm_dropout.summary(print_fn=lambda x: summary_buffer.write(x + '\n'))
model_summary_str = summary_buffer.getvalue()
summary_buffer.close()

# Write to file
with open(f"{MODEL_DIR}{model_name}_performance.txt", "w") as f:
    f.write(f"Model Name: {model_name}\n")
    f.write(f"Input Shape: {model_lstm_dropout.input_shape[1:]}\n")
    f.write(f"Total Parameters: {model_lstm_dropout.count_params() // 1000}K\n")
    f.write(f"Optimizer: Adam\n")
    f.write(f"Training Accuracy: {history.history['accuracy'][-1]:.4f}\n")
    f.write(f"Validation Accuracy: {history.history['val_accuracy'][-1]:.4f}\n")
    f.write("Model Summary:\n")
    f.write(model_summary_str)
```

```
Model Name: LSTM_Frozen_Embeddings_Additional_Dropout
```

```
/home/megarnol/projects/MSDS_Notes_Playground/.venv/lib/python3.10/site-
packages/keras/src/engine/training.py:3103: UserWarning: You are saving your
model as an HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

8.1 Results for LSTM with Additional Regularization

[165]:
```python
# Test dataset
# Predictions of test dataset
test_predictions, submissions = get_test_predictions(
    model=model_lstm_dropout,
    tokenizer=tokenizer,
    max_length=100,
    texts=df_test["cleaned_text"].tolist(),
    ids=df_test["id"].tolist(),
)
model_name = "LSTM_Frozen_Embeddings_Additional_Dropout"
test_predictions.to_csv(f"{DATA_DIR}/submissions/{model_name}_test_predictions.
 ↪csv", index=False)
submissions.to_csv(f"{DATA_DIR}/submissions/{model_name}_submission.csv",␣
 ↪index=False)
```

```
102/102 [==============================] - 10s 101ms/step
```

[175]:
```python
# Get train predictions
train_predictions, _ = get_test_predictions(
    model=model_lstm_dropout,
```

```
    tokenizer=tokenizer,
    max_length=100,
    texts=df_train["cleaned_text"].tolist(),
    ids=df_train["id"].tolist(),
)
model_name = "LSTM_Frozen_Embeddings_Additional_Dropout"
train_predictions["actual"] = df_train["target"].values
train_predictions.to_csv(f"{DATA_DIR}/submissions/
 ↪{model_name}_train_predictions.csv", index=False)
tp_samples, fp_samples, tn_samples, fn_samples =␣
 ↪get_tp_fp_tn_fn_samples(train_predictions)

print(f"Classification Report:\n")
print(classification_report(train_predictions["actual"],␣
 ↪train_predictions["target"]))
print(f"True Positive Tweet Samples: \n{tp_samples}\n")
print(f"False Positive Tweet Samples: \n{fp_samples}\n")
print(f"True Negative Tweet Samples: \n{tn_samples}\n")
print(f"False Negative Tweet Samples: \n{fn_samples}\n")
```

```
191/191 [==============================] - 20s 106ms/step
Classification Report:

              precision    recall  f1-score   support

           0       0.84      0.88      0.86      3473
           1       0.83      0.78      0.80      2617

    accuracy                           0.84      6090
   macro avg       0.83      0.83      0.83      6090
weighted avg       0.84      0.84      0.84      6090

True Positive Tweet Samples:
5477                               what the fuck is going on at trent bridge
reminds me of englands collapse out in the caribbean back in the s
3372    abe government said the missiles were not weapon so jsdf could provide
them to the ally when collective self defense right was exercised
1747
drought fuels bush fires in jamaica       re
Name: text, dtype: object

False Positive Tweet Samples:
3960                                   the advantages apropos of in flames
favorable regard mississauga ontario pwhvgwax
1813                  france agreed to repay russia for two warships which were
never delivered after economic sanctions
4598    ahrar al sham in our negotiations with iran over al zabadani they wanted
```

```
all sunnis evacuated out of al zabadani
Name: text, dtype: object

True Negative Tweet Samples:
381                                                    we can hear the
conversation now sorry senator we thought you said electrocute  million etc
3071    the riddler would be the best earlyexit primary presidential wannabe
ever all certain of his chances until he gets wrecked by a rich guy
302
drop it down on a nigga do damage
Name: text, dtype: object

False Negative Tweet Samples:
4333                                                    it was hella crazy
fights an ambulance and a couple mosh pits
1369
volcano bowl drink
2207    a look at state actions a year after fergusons upheaval  md is mentioned
in the last group for a reporting bill
Name: text, dtype: object
```

9 GRU Model

Gated Recurrent Units (GRU) are a type of recurrent neural network that is designed to help handle the vanishing gradient problem and capture more long-term information in sequential data. These networks are similar to LSTM networks, but have a more simple architecture.

There are two gates in GRUs network [4]. - The update gate determines if the cell state should be updated with new information or if it should retain the previous information. - The reset gate determines if the previous information should be forgotten or retained

These networks are more efficient to train compared to LSTMs, as they have fewer parameters, but their performance can be comparable. I will use this new architecture to see if it can outperform the comparable baseline LSTM model. In addition to changing the architecture, I will also experiment with fine-tuning the embedding layer and using a different embedding model (GLoVe).

As expected, the GRU model was able to perform similarly to the LSTM model, while being more efficient computationally.

The results of this model are: |Model Name| Training Accuracy | Training Precision | Training Recall | Validation Accuracy | Kaggle Test Accuracy | |—|—|-|-|—|—| |LSTM Baseline|0.83|0.85|0.82|0.81| 0.79619| |LSTM with Additional Dropout|0.84|0.83|0.83|0.81| 0.79129| |GRU Model|0.83|0.83|0.81|0.82| 0.79252|

[88]:
```
from tensorflow.keras import layers, models, optimizers, callbacks

# import sequential
from keras.models import Sequential
```

```python
vocab_size = len(tokenizer.word_index) + 1
assert (
    vocab_size == embedding_matrix.shape[0]
), "Vocab size and embedding matrix size do not match"

seq_len = train_padded.shape[1]  # 100
assert (
    train_padded.shape[1] == val_padded.shape[1]
), "Training and validation data sequence lengths do not match"

input_layer = layers.Input(shape=(seq_len,), dtype="int32")

embedding_layer_frozen = layers.Embedding(
    input_dim=vocab_size,
    output_dim=300,
    weights=[embedding_matrix],
    input_length=seq_len,
    mask_zero=True,  # Prevents RNN from updating on padded values
    trainable=False,  # Potential to unfreeze this later.
)

gru_layer = layers.Bidirectional(layers.GRU(64, return_sequences=False))

dropout_layer = layers.Dropout(0.3)

dense_layer = layers.Dense(16, activation="relu")


output_layer = layers.Dense(1, activation="sigmoid")

model_gru = models.Sequential(
    [
        input_layer,
        embedding_layer_frozen,
        gru_layer,
        dropout_layer,
        dense_layer,
        output_layer,
    ]
)

model_gru.compile(
    optimizer=optimizers.Adam(learning_rate=1e-4),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)
```

```
model_gru.summary()

early_stopping = callbacks.EarlyStopping(
    monitor="val_loss",
    patience=5,
    restore_best_weights=True,
)
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_6 (Embedding)     (None, 100, 300)          3591000

 bidirectional_6 (Bidirecti  (None, 128)               140544
 onal)

 dropout_3 (Dropout)         (None, 128)               0

 dense_12 (Dense)            (None, 16)                2064

 dense_13 (Dense)            (None, 1)                 17

=================================================================
Total params: 3733625 (14.24 MB)
Trainable params: 142625 (557.13 KB)
Non-trainable params: 3591000 (13.70 MB)
_________________________________________________________________
```

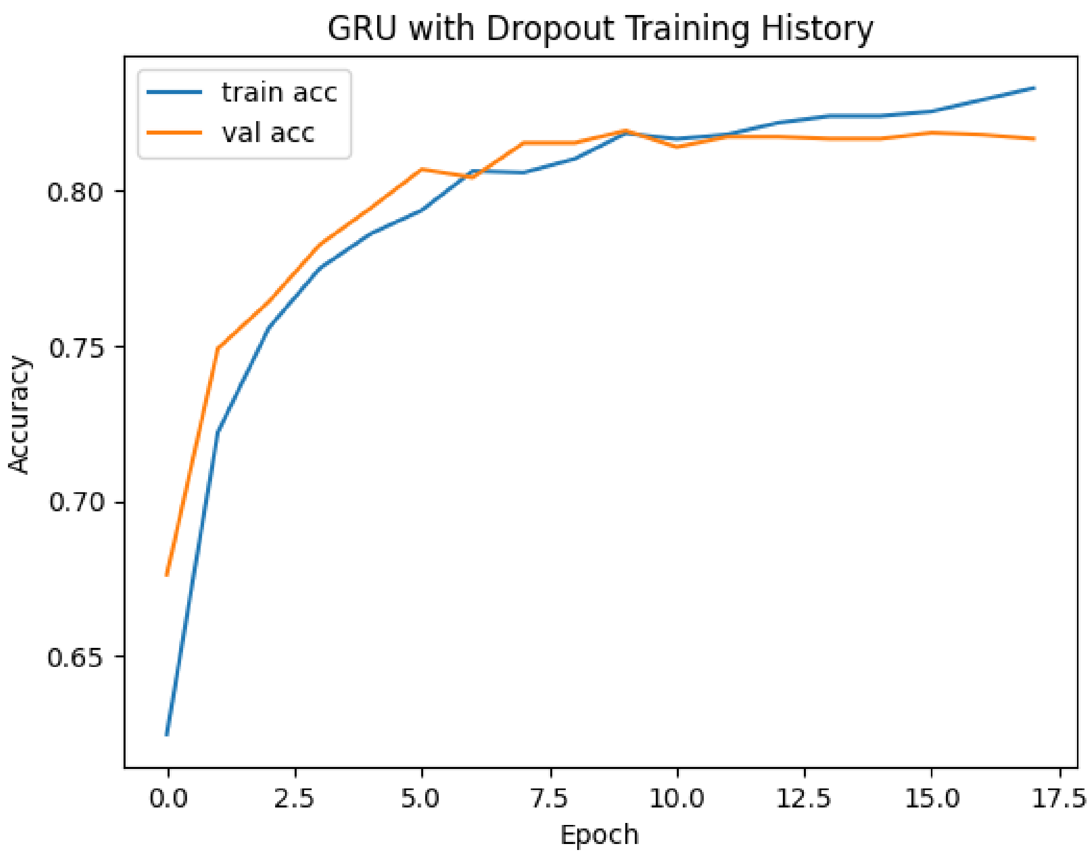
[ ]:
```
model_name = "GRU_Frozen_Embeddings"
if os.path.exists(f"{DATA_DIR}/models/{model_name}.h5"):
    model, history = load_model_and_history(model_name)
else:
    history = model.fit(
        train_padded,
        df_train["target"].values,
        validation_data=(val_padded, df_val["target"].values),
        epochs=30,
    batch_size=64,
    callbacks=[early_stopping],
)
```

```
Epoch 1/30
96/96 [==============================] - 15s 107ms/step - loss: 0.6574 -
accuracy: 0.6248 - val_loss: 0.6245 - val_accuracy: 0.6763
Epoch 2/30
```

```
96/96 [==============================] - 9s 94ms/step - loss: 0.5933 - accuracy:
0.7222 - val_loss: 0.5527 - val_accuracy: 0.7492
Epoch 3/30
96/96 [==============================] - 9s 95ms/step - loss: 0.5258 - accuracy:
0.7558 - val_loss: 0.4989 - val_accuracy: 0.7643
Epoch 4/30
96/96 [==============================] - 9s 90ms/step - loss: 0.4826 - accuracy:
0.7750 - val_loss: 0.4688 - val_accuracy: 0.7827
Epoch 5/30
96/96 [==============================] - 9s 94ms/step - loss: 0.4626 - accuracy:
0.7862 - val_loss: 0.4551 - val_accuracy: 0.7945
Epoch 6/30
96/96 [==============================] - 9s 89ms/step - loss: 0.4458 - accuracy:
0.7938 - val_loss: 0.4416 - val_accuracy: 0.8070
Epoch 7/30
96/96 [==============================] - 9s 91ms/step - loss: 0.4353 - accuracy:
0.8064 - val_loss: 0.4430 - val_accuracy: 0.8043
Epoch 8/30
96/96 [==============================] - 9s 91ms/step - loss: 0.4289 - accuracy:
0.8059 - val_loss: 0.4332 - val_accuracy: 0.8155
Epoch 9/30
96/96 [==============================] - 10s 103ms/step - loss: 0.4216 -
accuracy: 0.8103 - val_loss: 0.4307 - val_accuracy: 0.8155
Epoch 10/30
96/96 [==============================] - 9s 90ms/step - loss: 0.4155 - accuracy:
0.8186 - val_loss: 0.4300 - val_accuracy: 0.8194
Epoch 11/30
96/96 [==============================] - 9s 89ms/step - loss: 0.4142 - accuracy:
0.8167 - val_loss: 0.4337 - val_accuracy: 0.8142
Epoch 12/30
96/96 [==============================] - 9s 90ms/step - loss: 0.4101 - accuracy:
0.8182 - val_loss: 0.4287 - val_accuracy: 0.8175
Epoch 13/30
96/96 [==============================] - 9s 93ms/step - loss: 0.4072 - accuracy:
0.8220 - val_loss: 0.4268 - val_accuracy: 0.8175
Epoch 14/30
96/96 [==============================] - 9s 95ms/step - loss: 0.4042 - accuracy:
0.8241 - val_loss: 0.4295 - val_accuracy: 0.8168
Epoch 15/30
96/96 [==============================] - 9s 93ms/step - loss: 0.3984 - accuracy:
0.8241 - val_loss: 0.4288 - val_accuracy: 0.8168
Epoch 16/30
96/96 [==============================] - 9s 93ms/step - loss: 0.3965 - accuracy:
0.8256 - val_loss: 0.4269 - val_accuracy: 0.8188
Epoch 17/30
96/96 [==============================] - 9s 94ms/step - loss: 0.3937 - accuracy:
0.8294 - val_loss: 0.4270 - val_accuracy: 0.8181
Epoch 18/30
```

```
96/96 [==============================] - 9s 96ms/step - loss: 0.3895 - accuracy:
0.8332 - val_loss: 0.4277 - val_accuracy: 0.8168
```

[90]:
```
# Plot epoch history

plt.plot(history.history["accuracy"], label="train acc")
plt.plot(history.history["val_accuracy"], label="val acc")
plt.title("GRU with Dropout Training History")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



[ ]:
```
# Save the model
from io import StringIO
model_name = "GRU_Frozen_Embeddings"
print(f"Model Name: {model_name}")
model_gru.save(f"{MODEL_DIR}/{model_name}.h5")

import pickle
```

```
with open(f"{MODEL_DIR}/{model_name}_history.pkl", "wb") as f:
    pickle.dump(history.history, f)

# Capture model summary
summary_buffer = StringIO()
model_gru.summary(print_fn=lambda x: summary_buffer.write(x + '\n'))
model_summary_str = summary_buffer.getvalue()
summary_buffer.close()

# Write to file
with open(f"{MODEL_DIR}{model_name}_performance.txt", "w") as f:
    f.write(f"Model Name: {model_name}\n")
    f.write(f"Input Shape: {model_gru.input_shape[1:]}\n")
    f.write(f"Total Parameters: {model_gru.count_params() // 1000}K\n")
    f.write(f"Optimizer: Adam\n")
    f.write(f"Training Accuracy: {history.history['accuracy'][-1]:.4f}\n")
    f.write(f"Validation Accuracy: {history.history['val_accuracy'][-1]:.4f}\n")
    f.write("Model Summary:\n")
    f.write(model_summary_str)
```

```
Model Name: GRU_Frozen_Embeddings
```

```
/home/megarnol/projects/MSDS_Notes_Playground/.venv/lib/python3.10/site-
packages/keras/src/engine/training.py:3103: UserWarning: You are saving your
model as an HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

9.1 Results for GRU model

[164]:
```
# Test Predictions
# load GRU_Frozen_Embedding Model
from keras.models import load_model
model_path = f"{MODEL_DIR}GRU_Frozen_Embeddings.h5"
pre_finetune_gru_model = load_model(model_path)

test_predictions, submissions = get_test_predictions(
    model=pre_finetune_gru_model,
    tokenizer=tokenizer,
    max_length=100,
    texts=df_test["cleaned_text"].tolist(),
    ids=df_test["id"].tolist(),
)
model_name = "GRU_Frozen_Embeddings"
test_predictions.to_csv(f"{DATA_DIR}/submissions/{model_name}_test_predictions.
 ↪csv", index=False)
```

```
submissions.to_csv(f"{DATA_DIR}/submissions/{model_name}_submission.csv",␣
↪index=False)
```

102/102 [==============================] - 3s 18ms/step

[176]:

```
# Get train predictions
train_predictions, _ = get_test_predictions(
    model=pre_finetune_gru_model,
    tokenizer=tokenizer,
    max_length=100,
    texts=df_train["cleaned_text"].tolist(),
    ids=df_train["id"].tolist(),
)
model_name = "GRU_Frozen_Embeddings"
train_predictions["actual"] = df_train["target"].values
train_predictions.to_csv(f"{DATA_DIR}/submissions/
↪{model_name}_train_predictions.csv", index=False)
tp_samples, fp_samples, tn_samples, fn_samples =␣
↪get_tp_fp_tn_fn_samples(train_predictions)

print(f"Classification Report:\n")
print(classification_report(train_predictions["actual"],␣
↪train_predictions["target"]))
print(f"True Positive Tweet Samples: \n{tp_samples}\n")
print(f"False Positive Tweet Samples: \n{fp_samples}\n")
print(f"True Negative Tweet Samples: \n{tn_samples}\n")
print(f"False Negative Tweet Samples: \n{fn_samples}\n")
```

```
191/191 [==============================] - 2s 11ms/step
Classification Report:

              precision    recall  f1-score   support

           0       0.81      0.91      0.86      3473
           1       0.86      0.71      0.78      2617

    accuracy                           0.83      6090
   macro avg       0.83      0.81      0.82      6090
weighted avg       0.83      0.83      0.82      6090

True Positive Tweet Samples:
4758    obama declares disaster for typhoondevastated saipan obama signs
disaster declaration for northern marians a
1036                           sinking ships burning buildings amp falling
objects are what reminds me of the old us
1329                                                            obama declares
disaster for typhoondevastated saipan
```

```
Name: text, dtype: object

False Positive Tweet Samples:
1183                                   happening now  hatzolah ems ambulance
responding with dual sirens and
3955    this sale and demolition trend near metrotown is sure resulting in some
poorly maintained apartments
351                     over half of poll respondents worry nuclear disaster
fading from public consciousness
Name: text, dtype: object

True Negative Tweet Samples:
2280                             marketforce perth named winner of sirens round  for
iinet nbn buffering cat shark radio spot
4818                                                    guys to scared to show
his real name anyway he knows ill bomb him
36      alton brown just did a livestream and he burned the butter and touched
the hot plate too soon and made a nut joke
Name: text, dtype: object

False Negative Tweet Samples:
3949    montgomery come for the blazing hot weatherstay for the stds yet another
rejected city slogan
4129                                                      some people are really
natural disaster too
4291                                                               there might
be casualties tomorrow
Name: text, dtype: object
```

9.2 Update GRU model with fine-tuning the embedding layers

Since I am using a pretrained embedding model, I want to see if I can update the weights of the embedding layer to learn more about tweets specifically. This dataset typically contains slang and other informal language that may not be represented by the original embedding model.

To fine-tune the embedding layer, I will set the trainable parameter to True. This will unfreeze the weights and enable the epochs to update the embedding layer. I will decrease the learning rate to 0.00005 to attempt to prevent overfitting and large updates. I will then take the test dataset and submit the results to Kaggle to see if there is an improvement over the baseline GRU model.

GRU with Fine-tuned Embedding Layer Results:

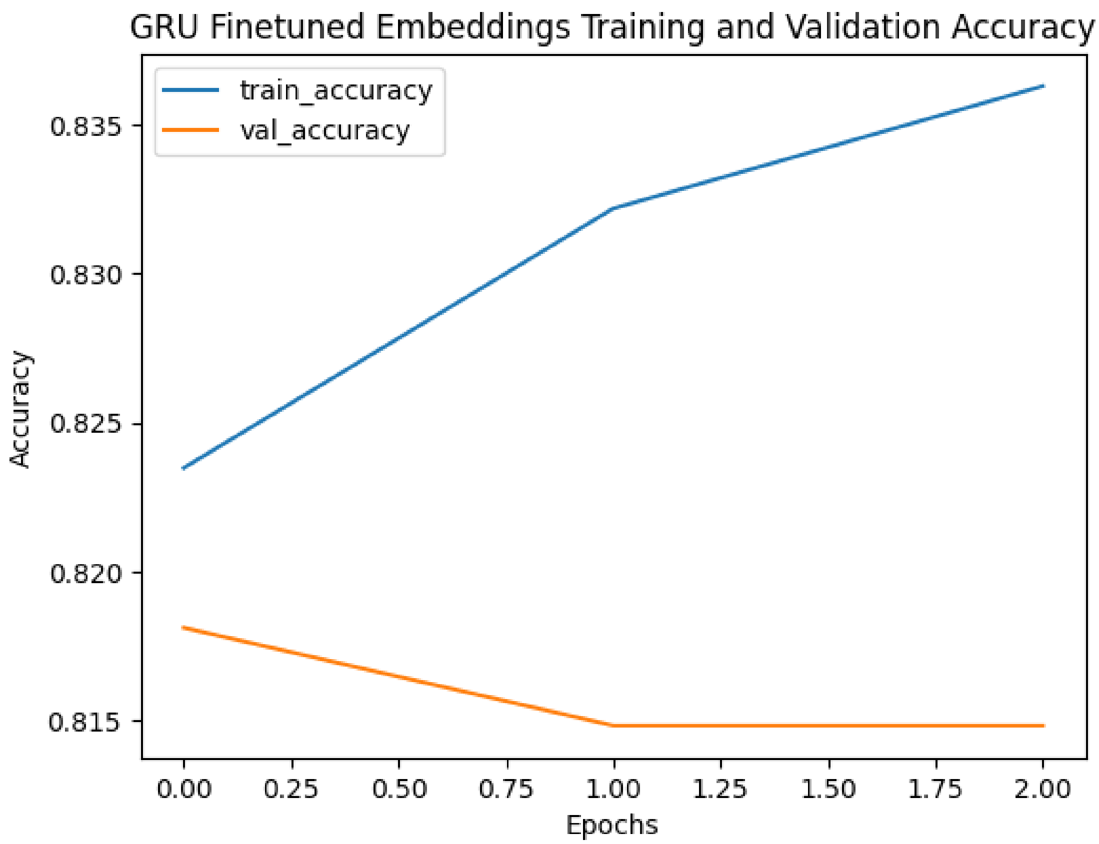
Model Name	Training Accuracy	Training Precision	Training Recall	Validation Accuracy	Kaggle Test Accuracy
LSTM Baseline	0.83	0.85	0.82	0.81	0.79619
LSTM with Additional Dropout	0.84	0.83	0.83	0.81	0.79129

Model Name	Training Accuracy	Training Precision	Training Recall	Validation Accuracy	Kaggle Test Accuracy
GRU Model	0.83	0.83	0.81	0.82	0.79252
GRU with Finetuned Embedding Layer	0.84	0.85	0.83	0.81	0.79497

[ ]:
```
model_name = "GRU_Embeddings_Finetuned"
if os.path.exists(f"{DATA_DIR}/models/{model_name}.h5"):
    model_gru, history = load_model_and_history(model_name)
else:
    # load original frozen model
    model_gru = tf.keras.models.load_model(f"{DATA_DIR}/models/
↪GRU_Frozen_Embeddings.h5")
    # Allow embedding layer to be trained
    model_gru.layers[0].trainable = True   # embedding is layer[0] in Sequential

    # Reduce learning rate
    model_gru.compile(
        optimizer=optimizers.Adam(learning_rate=5e-5),
        loss="binary_crossentropy",
        metrics=["accuracy"]
    )

    # Train the model
    history_finetune = model_gru.fit(
        train_padded,
        df_train["target"].values,
        validation_data=(val_padded, df_val["target"].values),
        epochs=3,            # usually 1-3 is enough for fine-tuning
        batch_size=64,
        callbacks=[early_stopping],
        verbose=1
    )
```

```
Epoch 1/3
96/96 [==============================] - 21s 170ms/step - loss: 0.4016 -
accuracy: 0.8235 - val_loss: 0.4269 - val_accuracy: 0.8181
Epoch 2/3
96/96 [==============================] - 12s 126ms/step - loss: 0.3898 -
accuracy: 0.8322 - val_loss: 0.4284 - val_accuracy: 0.8148
Epoch 3/3
96/96 [==============================] - 11s 113ms/step - loss: 0.3791 -
accuracy: 0.8363 - val_loss: 0.4231 - val_accuracy: 0.8148
```

[94]:
```
plt.plot(history_finetune.history["accuracy"], label="train_accuracy")
plt.plot(history_finetune.history["val_accuracy"], label="val_accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.title("GRU Finetuned Embeddings Training and Validation Accuracy")
plt.show()
```



[ ]:
```
# Save the model
from io import StringIO
model_name = "GRU_Embeddings_Finetuned"
print(f"Model Name: {model_name}")
model_gru.save(f"{MODEL_DIR}/{model_name}.h5")

import pickle

with open(f"{MODEL_DIR}/{model_name}_history.pkl", "wb") as f:
    pickle.dump(history.history, f)

# Capture model summary
```

```
summary_buffer = StringIO()
model_gru.summary(print_fn=lambda x: summary_buffer.write(x + '\n'))
model_summary_str = summary_buffer.getvalue()
summary_buffer.close()

# Write to file
with open(f"{MODEL_DIR}{model_name}_performance.txt", "w") as f:
    f.write(f"Model Name: {model_name}\n")
    f.write(f"Input Shape: {model_gru.input_shape[1:]}\n")
    f.write(f"Total Parameters: {model_gru.count_params() // 1000}K\n")
    f.write(f"Optimizer: Adam\n")
    f.write(f"Training Accuracy: {history_finetune.history['accuracy'][-1]:.
↪4f}\n")
    f.write(f"Validation Accuracy: {history_finetune.
↪history['val_accuracy'][-1]:.4f}\n")
    f.write("Model Summary:\n")
    f.write(model_summary_str)
```

```
Model Name: GRU_Embeddings_Finetuned
```

9.3 Results for GRU model with Finetuned Embedding Layer

[163]:
```
# Predictions of test dataset
test_predictions, submissions = get_test_predictions(
    model=model_gru,
    tokenizer=tokenizer,
    max_length=100,
    texts=df_test["cleaned_text"].tolist(),
    ids=df_test["id"].tolist(),
)
model_name = "GRU_Embeddings_Finetuned"
test_predictions.to_csv(f"{DATA_DIR}/submissions/{model_name}_test_predictions.
↪csv", index=False)
submissions.to_csv(f"{DATA_DIR}/submissions/{model_name}_submission.csv",␣
↪index=False)
```

```
102/102 [==============================] - 2s 17ms/step
```

[177]:
```
# Get train predictions
train_predictions, _ = get_test_predictions(
    model=model_gru,
    tokenizer=tokenizer,
    max_length=100,
    texts=df_train["cleaned_text"].tolist(),
    ids=df_train["id"].tolist(),
)
model_name = "GRU_Embeddings_Finetuned"
```

```
train_predictions["actual"] = df_train["target"].values
train_predictions.to_csv(f"{DATA_DIR}/submissions/
↪{model_name}_train_predictions.csv", index=False)
tp_samples, fp_samples, tn_samples, fn_samples =␣
↪get_tp_fp_tn_fn_samples(train_predictions)

print(f"Classification Report:\n")
print(classification_report(train_predictions["actual"],␣
↪train_predictions["target"]))
print(f"True Positive Tweet Samples: \n{tp_samples}\n")
print(f"False Positive Tweet Samples: \n{fp_samples}\n")
print(f"True Negative Tweet Samples: \n{tn_samples}\n")
print(f"False Negative Tweet Samples: \n{fn_samples}\n")
```

```
191/191 [==============================] - 2s 11ms/step
Classification Report:

              precision    recall  f1-score   support

           0       0.82      0.92      0.87      3473
           1       0.88      0.74      0.80      2617

    accuracy                           0.84      6090
   macro avg       0.85      0.83      0.84      6090
weighted avg       0.85      0.84      0.84      6090

True Positive Tweet Samples:
2146
because if it were on fire thatd be a safety hazard
2866    iof murdered over  palestinian children  under  during gaza massacre
where was zionist moralityzionism is a world evil
5952
terrorist shot dead
Name: text, dtype: object

False Positive Tweet Samples:
2288              no dont evacuate the students just throw them in the dungeon
that is stupid
351     over half of poll respondents worry nuclear disaster fading from public
consciousness
3558                                                      evildead  annihilation of
civilization
Name: text, dtype: object

True Negative Tweet Samples:
1595          god the  are so cocky right now and i love it uribe obliterated
that ball then strutted the fuck out of the batters box
```

```
2320                                                    okay maybe not as extreme as
thunder and lightning but pretty much all other types
4420    sure i just burned about  calories after eating a giant bowl of mac and
cheese so i totally earned this  calorie klondike bar
Name: text, dtype: object

False Negative Tweet Samples:
2153       happy boy to mass murderer
1305        cindy noonanheartbreak in
1006    thunder buddys thunder buddys
Name: text, dtype: object
```

9.4 GRU with glove-twitter-200 embedding matrix

As the next model to test, I wanted to get a more targeted embedding model. The GLoVe[3] embedding model had twitter data in the training set, so I want to see if this improves the model's ability to understand the style of tweets. The GLoVe model has a dimension of 200, which is lower than the Word2Vec model's 300 dimensions. This reduction in dimensions could impact the model's performance, but I'm hoping that the more targeted training data will improve the model's ability to classify tweets.

This model turned out to be the worse performing model, despite the more targeted embedding model. However, it may be a result of the embedding vector having length 200 instead of the original 300. There may be some information loss by reducing the dimensions of the embedding vectors.

GRU with GLoVe Twitter Embedding Results:

Model Name	Training Accuracy	Training Precision	Training Recall	Validation Accuracy	Kaggle Test Accuracy
LSTM Baseline	0.83	0.85	0.82	0.81	0.79619
LSTM with Additional Dropout	0.84	0.83	0.83	0.81	0.79129
GRU Model	0.83	0.83	0.81	0.82	0.79252
GRU with Finetuned Embedding Layer	0.84	0.85	0.83	0.81	0.79497
GRU with GLoVe Twitter Embedding	0.83	0.83	0.82	0.80	0.79098

[96]:
```python
# Different Embedding model
import gensim.downloader as api

# if embedding_matrix doesn't exist
```

```
if not os.path.exists(f"{DATA_DIR}/embedding_matrix_glove-twitter.npy"):
    gt200 = api.load("glove-twitter-200")

    ## Map the word index to the embedding matrix
    embedding_dim_gt200 = 200
    embedding_matrix_gt200 = np.zeros((len(tokenizer.word_index) + 1,␣
 ↪embedding_dim_gt200))

    for word, i in tokenizer.word_index.items():
        embedding_vector = gt200[word] if word in gt200 else None
        if embedding_vector is not None:
            embedding_matrix_gt200[i] = embedding_vector
    # Save the embedding matrix
    np.save(f"{DATA_DIR}/embedding_matrix_glove-twitter.npy",␣
 ↪embedding_matrix_gt200)
else:
    embedding_matrix_gt200 = np.load(f"{DATA_DIR}/
 ↪embedding_matrix_glove-twitter.npy")

# Verify embedding matrix
print("Expected shape of embedding matrix:", (len(tokenizer.word_index) + 1,␣
 ↪200))
print("Embedding matrix shape:", embedding_matrix_gt200.shape)
```

```
[==================================================] 100.0% 758.5/758.5MB
downloaded
Expected shape of embedding matrix: (11970, 200)
Embedding matrix shape: (11970, 200)
```

[97]:

```
# GRU with glove-twitter-200 embedding matrix
vocab_size = len(tokenizer.word_index) + 1
assert (
    vocab_size == embedding_matrix_gt200.shape[0]
), "Vocab size and embedding matrix size do not match"

seq_len = train_padded.shape[1]  # 100
assert (
    train_padded.shape[1] == val_padded.shape[1]
), "Training and validation data sequence lengths do not match"

input_layer = layers.Input(shape=(seq_len,), dtype="int32")

embedding_layer_frozen = layers.Embedding(
    input_dim=vocab_size,
    output_dim=200,
    weights=[embedding_matrix_gt200],
    input_length=seq_len,
```

```
    mask_zero=True,  # Prevents RNN from updating on padded values
    trainable=False,  # Potential to unfreeze this later.
)

gru_layer = layers.Bidirectional(layers.GRU(64, return_sequences=False))

dropout_layer = layers.Dropout(0.3)

dense_layer = layers.Dense(16, activation="relu")

output_layer = layers.Dense(1, activation="sigmoid")

model_gru_glove = models.Sequential(
    [
        input_layer,
        embedding_layer_frozen,
        gru_layer,
        dropout_layer,
        dense_layer,
        output_layer,
    ]
)

model_gru_glove.compile(
    optimizer=optimizers.Adam(learning_rate=1e-4),
    loss="binary_crossentropy",
    metrics=["accuracy"],
)

model_gru_glove.summary()

early_stopping = callbacks.EarlyStopping(
    monitor="val_loss",
    patience=5,
    restore_best_weights=True,
)
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_7 (Embedding)     (None, 100, 200)          2394000

 bidirectional_7 (Bidirecti  (None, 128)               102144
 onal)

 dropout_4 (Dropout)         (None, 128)               0
```

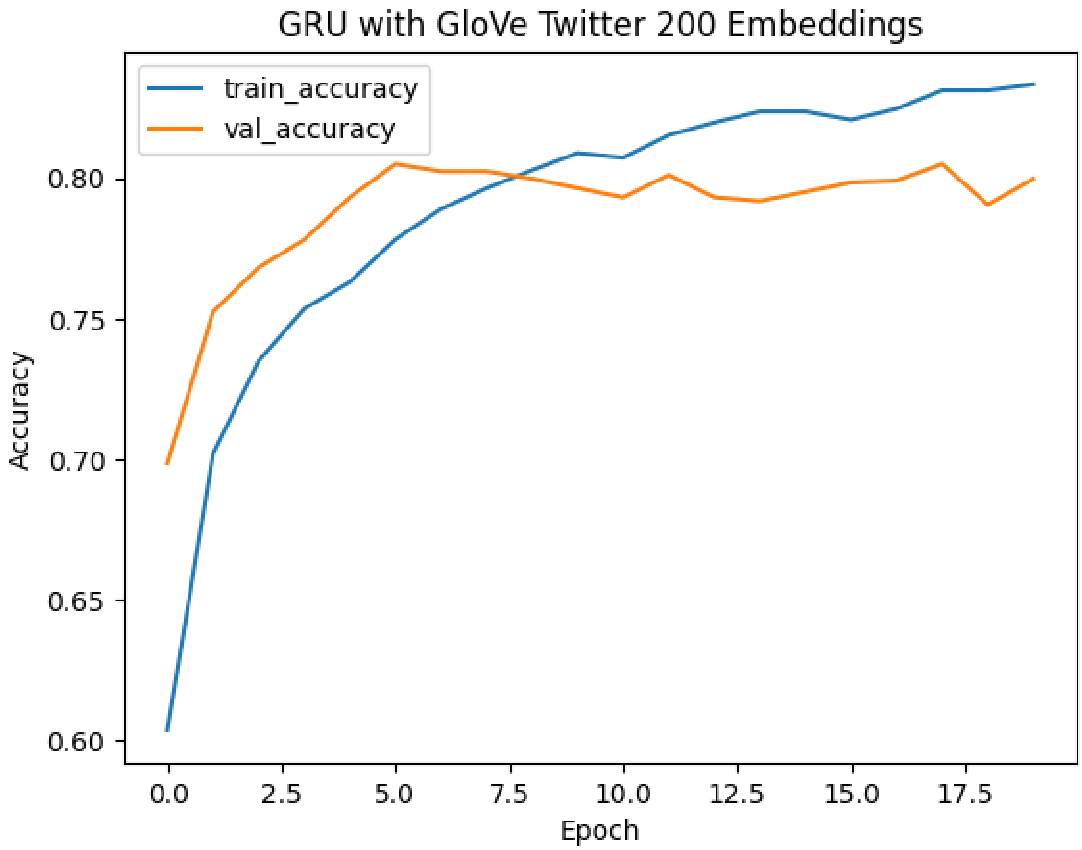
```
 dense_14 (Dense)            (None, 16)                2064

 dense_15 (Dense)            (None, 1)                 17

=================================================================
Total params: 2498225 (9.53 MB)
Trainable params: 104225 (407.13 KB)
Non-trainable params: 2394000 (9.13 MB)
_________________________________________________________________
```

[98]:
```python
history = model_gru_glove.fit(
    train_padded,
    df_train["target"].values,
    validation_data=(val_padded, df_val["target"].values),
    epochs=30,
    batch_size=64,
    callbacks=[early_stopping],
)
```

```
Epoch 1/30
96/96 [==============================] - 16s 103ms/step - loss: 0.6600 -
accuracy: 0.6038 - val_loss: 0.6043 - val_accuracy: 0.6986
Epoch 2/30
96/96 [==============================] - 8s 84ms/step - loss: 0.5888 - accuracy:
0.7020 - val_loss: 0.5390 - val_accuracy: 0.7525
Epoch 3/30
96/96 [==============================] - 8s 84ms/step - loss: 0.5423 - accuracy:
0.7350 - val_loss: 0.5046 - val_accuracy: 0.7682
Epoch 4/30
96/96 [==============================] - 8s 83ms/step - loss: 0.5130 - accuracy:
0.7535 - val_loss: 0.4804 - val_accuracy: 0.7781
Epoch 5/30
96/96 [==============================] - 8s 85ms/step - loss: 0.4875 - accuracy:
0.7631 - val_loss: 0.4613 - val_accuracy: 0.7932
Epoch 6/30
96/96 [==============================] - 8s 86ms/step - loss: 0.4709 - accuracy:
0.7782 - val_loss: 0.4479 - val_accuracy: 0.8050
Epoch 7/30
96/96 [==============================] - 8s 83ms/step - loss: 0.4551 - accuracy:
0.7890 - val_loss: 0.4402 - val_accuracy: 0.8024
Epoch 8/30
96/96 [==============================] - 8s 87ms/step - loss: 0.4437 - accuracy:
0.7964 - val_loss: 0.4366 - val_accuracy: 0.8024
Epoch 9/30
96/96 [==============================] - 7s 77ms/step - loss: 0.4351 - accuracy:
0.8028 - val_loss: 0.4341 - val_accuracy: 0.7997
Epoch 10/30
```

```
96/96 [==============================] - 9s 90ms/step - loss: 0.4262 - accuracy:
0.8089 - val_loss: 0.4352 - val_accuracy: 0.7965
Epoch 11/30
96/96 [==============================] - 8s 78ms/step - loss: 0.4221 - accuracy:
0.8072 - val_loss: 0.4417 - val_accuracy: 0.7932
Epoch 12/30
96/96 [==============================] - 8s 88ms/step - loss: 0.4163 - accuracy:
0.8153 - val_loss: 0.4311 - val_accuracy: 0.8011
Epoch 13/30
96/96 [==============================] - 8s 87ms/step - loss: 0.4108 - accuracy:
0.8197 - val_loss: 0.4302 - val_accuracy: 0.7932
Epoch 14/30
96/96 [==============================] - 8s 88ms/step - loss: 0.4066 - accuracy:
0.8236 - val_loss: 0.4467 - val_accuracy: 0.7919
Epoch 15/30
96/96 [==============================] - 8s 75ms/step - loss: 0.4042 - accuracy:
0.8236 - val_loss: 0.4296 - val_accuracy: 0.7951
Epoch 16/30
96/96 [==============================] - 8s 89ms/step - loss: 0.4028 - accuracy:
0.8207 - val_loss: 0.4300 - val_accuracy: 0.7984
Epoch 17/30
96/96 [==============================] - 8s 76ms/step - loss: 0.3993 - accuracy:
0.8246 - val_loss: 0.4298 - val_accuracy: 0.7991
Epoch 18/30
96/96 [==============================] - 7s 77ms/step - loss: 0.3947 - accuracy:
0.8312 - val_loss: 0.4309 - val_accuracy: 0.8050
Epoch 19/30
96/96 [==============================] - 8s 86ms/step - loss: 0.3907 - accuracy:
0.8312 - val_loss: 0.4391 - val_accuracy: 0.7905
Epoch 20/30
96/96 [==============================] - 8s 85ms/step - loss: 0.3878 - accuracy:
0.8333 - val_loss: 0.4306 - val_accuracy: 0.7997
```

[99]:
```
plt.plot(history.history["accuracy"], label="train_accuracy")
plt.plot(history.history["val_accuracy"], label="val_accuracy")
plt.title("GRU with GloVe Twitter 200 Embeddings")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```



[ ]:
```
# Save the model
from io import StringIO
model_name = "GRU_Glove-Twitter-200"
print(f"Model Name: {model_name}")
model_gru_glove.save(f"{MODEL_DIR}/{model_name}.h5")

import pickle

with open(f"{MODEL_DIR}/{model_name}_history.pkl", "wb") as f:
    pickle.dump(history.history, f)

# Capture model summary
summary_buffer = StringIO()
model_gru_glove.summary(print_fn=lambda x: summary_buffer.write(x + '\n'))
model_summary_str = summary_buffer.getvalue()
summary_buffer.close()

# Write to file
with open(f"{MODEL_DIR}{model_name}_performance.txt", "w") as f:
    f.write(f"Model Name: {model_name}\n")
```

```
    f.write(f"Input Shape: {model_gru_glove.input_shape[1:]}\n")
    f.write(f"Total Parameters: {model_gru_glove.count_params() // 1000}K\n")
    f.write(f"Optimizer: Adam\n")
    f.write(f"Training Accuracy: {history.history['accuracy'][-1]:.4f}\n")
    f.write(f"Validation Accuracy: {history.history['val_accuracy'][-1]:.4f}\n")
    f.write("Model Summary:\n")
    f.write(model_summary_str)
```

```
Model Name: GRU_Glove-Twitter-200
```

```
/home/megarnol/projects/MSDS_Notes_Playground/.venv/lib/python3.10/site-
packages/keras/src/engine/training.py:3103: UserWarning: You are saving your
model as an HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(
```

9.5 Results of GRU with twitter trained embedding model

[162]:
```
# Predictions of test dataset

test_predictions, submissions = get_test_predictions(
    model=model_gru_glove,
    tokenizer=tokenizer,
    max_length=100,
    texts=df_test["cleaned_text"].tolist(),
    ids=df_test["id"].tolist(),
)
model_name = "GRU_Glove-Twitter-200"
test_predictions.to_csv(f"{DATA_DIR}/submissions/{model_name}_test_predictions.
 ↪csv", index=False)
submissions.to_csv(f"{DATA_DIR}/submissions/{model_name}_submission.csv",␣
 ↪index=False)
```

```
102/102 [==============================] - 2s 18ms/step
```

[179]:
```
# Get train predictions
train_predictions, _ = get_test_predictions(
    model=model_gru_glove,
    tokenizer=tokenizer,
    max_length=100,
    texts=df_train["cleaned_text"].tolist(),
    ids=df_train["id"].tolist(),
)
model_name = "GRU_Glove-Twitter-200"
train_predictions["actual"] = df_train["target"].values
train_predictions.to_csv(f"{DATA_DIR}/submissions/
 ↪{model_name}_train_predictions.csv", index=False)
```

```
tp_samples, fp_samples, tn_samples, fn_samples =␣
 ↪get_tp_fp_tn_fn_samples(train_predictions)

print(f"Classification Report:\n")
print(classification_report(train_predictions["actual"],␣
 ↪train_predictions["target"]))
print(f"True Positive Tweet Samples: \n{tp_samples}\n")
print(f"False Positive Tweet Samples: \n{fp_samples}\n")
print(f"True Negative Tweet Samples: \n{tn_samples}\n")
print(f"False Negative Tweet Samples: \n{fn_samples}\n")
```

```
191/191 [==============================] - 2s 12ms/step
Classification Report:

              precision    recall  f1-score   support

           0       0.81      0.91      0.86      3473
           1       0.86      0.73      0.79      2617

    accuracy                           0.83      6090
   macro avg       0.83      0.82      0.82      6090
weighted avg       0.83      0.83      0.83      6090

True Positive Tweet Samples:
5835    dtn brazil refugio oil spill may have been costlier bigger than
projected a plains all american pipeline oi
3692                                                                japan
aogashima volcano by unknown  check it out
5038      years after atomic bombs japan still struggles with war past the
anniversary of the devastation wrought b
Name: text, dtype: object

False Positive Tweet Samples:
129              chinas stock market crash are there gems in the rubble
5264    my  this morning heading out to make a whirlwind trip down south
4655                  where will the winds take my gypsy blood this time
Name: text, dtype: object

True Negative Tweet Samples:
1495                                 to ouvindo sleeping with sirens awn
3284    i would say im dead but im not that right there was obliteration
2473                              yo i got bars and im not even a rapper
Name: text, dtype: object

False Negative Tweet Samples:
1073    breaking news tonight kids were rescued from play room after a week with
no food or water do to parents sex life haha
```

```
3491
well so much for outdoor postering
3958                                            and when will you be commenting on
ian taylors dealings with mass  murderer arkan
Name: text, dtype: object
```

10 Results and Classification Report

Overall, the models performed similarly, with the LSTM baseline model being the best, with the GRU with fine-tuned embedding layer being a close second. The GRU model was also more computationally efficient to train. I would recommend using the GRU architecture for future work, as it is more efficient to train and has similar performance metrics.

I also tried a completely different embedding model, GLoVe, which had some twitter data in the training set; however, I think the reduction in the embedding model's complexity (200 dimensions vs 300 dimensions) led to a decrease in performance.

Below are the metrics for the different models. The test accuracy was from the Kaggle submission, and the training metrics were from the confusion matrix on the dataset

Model Name	Training Accuracy	Training Precision	Training Recall	Validation Accuracy	Kaggle Test Accuracy
LSTM Baseline	0.83	0.85	0.82	0.81	0.79619
LSTM with Additional Dropout	0.84	0.83	0.83	0.81	0.79129
GRU Model	0.83	0.83	0.81	0.82	0.79252
GRU with Finetuned Embedding Layer	0.84	0.85	0.83	0.81	0.79497
GRU with GLoVe Twitter Embedding	0.83	0.83	0.82	0.80	0.79098

11 Discussion

Below are some questions and answers regarding the results of the model and what I learned from the project:

- **Why did fine-tuning the embedding model create a worse validation accuracy?**
  - I believe that fine-tuning the embedding layer caused the model to overfit on the training dataset. Despite this overfitting, the test accuracy on Kaggle was slightly better than the baseline GRU model. To prevent overfitting in the future, I would try to reduce the learning rate or add additioonal regularization or dropout to the original GRU model, prior to the fine-tuning.
- **Did the GLoVe embedding model improve performance?**

- Despite the GLoVe embedding model having tweets included in the dataset, this didn't translate to an improvement in the performance metrics. The GLoVe model has a dimension of 200, which is lower than the Word2Vec model's 300 dimensions. This reduction in dimensions could explain the decrease in performance despite the more targeted training dataset.

- **Why did LSTM model with similar architecture to the GRU model take more time?**
  - The LSTM model has more parameters to train compared to the GRU model, which has a more simple architecture. This increase in parameters will lead to longer training times. However, we can see that the GRU model has similar Kaggle performance accuracy to the LSTM model. I would recommend using the GRU architecture for future work, as it is more efficient to train and has similar performance metrics.
- **Why did the Additional Dropout not improve performance?**
  - The additional dropout in the LSTM layer helped with the overfitting, but it didn't translate into an overall improvement of the model's performance. It's possible that the dropout rate is too high and this could be tuned in future iterations.

12 Conclusion

Overall, the models performed similarly, despite the different architectures, fine-tunning of the embedding layers, and different embedding models. I think the original LSTM and GRU architectures were sufficient to capture the information in the tweets, as the test accuracies were all around 79-80%. The GRU model was more efficient to train, so I would recommend using that architecture for future work.

Future Work: - Do additional hyperparameter tuning to the LSTM's dropout and recurrent dropout rates. The values of 0.2 may be too high and negatively impactec the model's performance. - Experiment with different optimizers. I only used Adam, but there may be others that could improve the model's performance. - Twitter data is notoriously messy, so more advanced preprocessing could be done to help improve the model's performance. I just removed URLS, punctuation, special characters, and stop words; however, maybe there was some information loss by removing emojis or urls.

13 References

[1] "Natural Language Processing with Disaster Tweets," @kaggle, 2025. https://www.kaggle.com/competitions/nlp-getting-started/overview (accessed Aug. 23, 2025).

[2] "15.1. Word Embedding (word2vec) — Dive into Deep Learning 1.0.3 documentation," D2l.ai, 2025. https://d2l.ai/chapter_natural-language-processing-pretraining/word2vec.html (accessed Sep. 05, 2025).

[3] J. Pennington, "GloVe: Global Vectors for Word Representation," Stanford.edu, 2024. https://nlp.stanford.edu/projects/glove/ (accessed Sep. 05, 2025).

[4] Hemanth Pedamallu, "RNN vs GRU vs LSTM," Medium, Nov. 14, 2020. https://medium.com/analytics-vidhya/rnn-vs-gru-vs-lstm-863b0b7b1573 (accessed Sep. 14, 2025).