NARRATIVE ANALYSIS OF OPEN-SOURCE SOCIAL MEDIA ACTIVITY IN THE INDOPACOM AOR

THESIS

Aaron Glenn, Captain, USAF

AFIT-ENS-MS-22-M-131

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENS-MS-22-M-131

NARRATIVE ANALYSIS OF OPEN-SOURCE SOCIAL MEDIA ACTIVITY IN THE INDOPACOM AOR

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Operations Research

Aaron Glenn, B.S.

Captain, USAF

March 24, 2022

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENS-MS-22-M-131

NARRATIVE ANALYSIS OF OPEN-SOURCE SOCIAL MEDIA ACTIVITY IN THE INDOPACOM AOR

THESIS

Aaron Glenn, B.S.
Captain, USAF

Committee Membership:

LTC Phillip M. LaCasse, Ph.D
Chair

Dr. Bruce A. Cox
Reader

AFIT-ENS-MS-22-M-131

Abstract

Emotion classification can be a powerful tool to derive narratives from social media data. Traditional machine learning models that perform emotion classification on Indonesian Twitter data exist but rely on closed-source features. Recurrent Neural Networks can meet or exceed the performance of state-of-the-art traditional machine learning techniques using exclusively open-source data and models. Specifically, these results show that Recurrent Neural Network variants can produce more than an 8% gain in accuracy in comparison to Logistic Regression and SVM techniques and a 15% gain over Random Forest when using FastText embeddings. This research found a statistical significance in the performance of a single layer Bi-directional Long Short-Term Memory model over a 2-layer stacked Bi-directional Long Short-Term Memory model. This research also found that a single layer Bi-directional Long Short-Term Memory Recurrent Neural Network met the performance of a state-of-the-art Logistic Regression model with supplemental closed-source features from a study by Saputri et al. (2018) when classifying the emotion of Indonesian Tweets. This model can be provided to operational units within the INDOPACOM theater giving them the ability to identify social media posts based on predicted emotion class - allowing them to gauge public reaction to military exercises in theater.

Contents

Page

Abstract iv

List of Figures vii

List of Tables ix

I. Introduction 1

1.1 General Issue 1
1.2 Problem Statement 2
1.3 Research Objectives 2
1.4 Organization 4

II. Literature Review 5

2.1 Information Warfare 5
2.2 Sentiment Analysis 7
2.3 Emotion Classification 8
2.4 Recurrent Neural Network Variants 10
2.5 Limitations 13
2.6 Primary Sources 16

III. Methodology 17

3.1 Artificial Neural Networks 17
3.2 Deep Neural Networks 20
3.3 Recurrent Neural Networks 23
3.3.1 LSTM and GRU 26
3.4 Word Embeddings 29
3.5 Dataset 30
3.6 Evaluation Metrics 32
3.7 Model Architecture 34

IV. Results and Analysis 37

4.1 Programming Platform 37
4.2 Cross Validation Results 37
4.3 Time Trials 39
4.4 Statistical Significance 40
4.5 Hyperparameter Tuning 41
4.6 Model Examination 44

Page

V. Conclusions 49

5.1 Future Work 50

Appendix A. Model Code 51

Appendix B. Aggregate Cross Validation Analysis 63

Bibliography 66

List of Figures

Figure Page

1. ANNs performing simple logical computations (Reprinted, with permission, from Géron, 2019 © O'Reilly) . . . 17

2. Threshold Logic Unit (Reprinted, with permission, from Géron, 2019 © O'Reilly) . . . 18

3. Perceptron Diagram (Reprinted, with permission, from Géron, 2019 © O'Reilly) . . . 19

4. Multi-Layer Perceptron (Reprinted, with permission, from Géron, 2019 © O'Reilly) . . . 20

5. Logistic sigmoid activation saturation (Reprinted, with permission, from Géron, 2019 © O'Reilly) . . . 21

6. ReLU activation function . . . 22

7. Dropout in a MLP (Reprinted, with permission, from Géron, 2019 © O'Reilly) . . . 22

8. A recurrent neuron (left), unrolled through time (right) (Reprinted, with permission, from Géron, 2019 © O'Reilly) . . . 24

9. A layer of recurrent neurons (left), unrolled through time (right) (Reprinted, with permission, from Géron, 2019 © O'Reilly) . . . 24

10. A simple memory cell (Reprinted, with permission, from Géron, 2019 © O'Reilly) . . . 25

11. LSTM Cell (Reprinted, with permission, from Géron, 2019 © O'Reilly) . . . 26

12. GRU Cell (Reprinted, with permission, from Géron, 2019 © O'Reilly) . . . 27

13. Bidirectional Recurrent Neural Network (Reprinted, with permission, from Xiao and Liang, 2016 © Springer) . . . 28

14. 2-Layer Stacked Bi-LSTM (Reprinted, with permission, from Zhou et al. © 2018 IEEE) . . . 28

Figure Page

15. Classification Balance of Tweets 31

16. FastText RNN Variant Cross Validation Results 39

17. FastText RNN Variant Bootstrap Results 40

18. Single-Layer Bi-LSTM Model Accuracy 44

19. Single-Layer Bi-LSTM Model Loss 44

20. Single-Layer Bi-LSTM Confusion Matrix 46

21. Aggregate Cross Validation LSTM Confusion Matrix 63

22. Aggregate Cross Validation Bi-LSTM Confusion Matrix 64

23. Aggregate Cross Validation Stacked Bi-LSTM Confusion Matrix 64

24. Aggregate Cross Validation GRU Confusion Matrix 65

List of Tables

Table		Page
1.	Model Architecture (Reprinted, with permission, from Devi et al. © 2021 IEEE)	12
2.	Comparative Macro-Level F1 Scores (Reprinted, with permission, from Saputri et al. © 2018 IEEE)	33
3.	LSTM/GRU/Bi-LSTM Models in Keras	35
4.	Stacked Bi-LSTM Model in Keras	36
5.	Embedding Technique Comparison (Saputri et al., 2018)	37
6.	Comparison Using All Features (Saputri et al., 2018)	38
7.	FastText Time Trials	39
8.	10-Sample Bootstrap Macro-Level F1 Scores	40
9.	Cell Unit Tuning Macro-Level F1 Scores (Saputri et al., 2018)	42
10.	Batch Size Tuning Macro-Level F1 Scores (Saputri et al., 2018)	42
11.	Final Single-Layer Bi-LSTM Model in Keras	43
12.	Classification Report (Glenn, 2022)	45
13.	Classification Report (Reprinted, with permission, from Saputri et al. ©2018 IEEE)	45
14.	Neutral Post Classification	48

NARRATIVE ANALYSIS OF OPEN-SOURCE SOCIAL MEDIA ACTIVITY IN THE INDOPACOM AOR

I. Introduction

1.1 General Issue

For over a decade, experts have argued that the U.S. Military should "shift strategic centers of gravity from the physical to the human aspects of warfare" (Gavrilis, 2009). This philosophy is now widely accepted in the context of small-scale operations (Beskow and Carley, 2019). As adversaries have demonstrated an exceptional ability to influence narratives through the medium of social media, the Department of Defense has acknowledged that the modern security environment has become "increasingly complex" and "defined by rapid technological change" with "challenges from adversaries in every operating domain" (Nimmo, 2015; Department of Defense, 2018). This new operating environment requires that U.S. forces increase their situational awareness of social media activity in theater.

Sentiment analysis has become an increasingly popular field for research with the increasing preeminence of the internet and social media. Twitter, in particular, has become a lightning rod for researchers aiming to model human language through various machine learning techniques. Notably, most of this research has been performed in the English-speaking world - leaving other languages relatively untouched and ready for discovery (Saputri et al., 2018).

1.2 Problem Statement

In July 2020, an operational unit within INDOPACOM was allocated personnel for dedicated collection and analysis of Open-Source Intelligence (OSINT) Data. The unit quickly realized it lacked the tools necessary to ingest and derive insights from large scale social media data. The processes at the time were user intensive and relied heavily on manual operations (Smithmeyer, 2021).

The operational unit initially requested assistance from The Research and Analysis Center-Monterey (TRAC-MTRY), which collaborated with partners at the Naval Postgraduate School (NPS) to create an R-Shiny tool for OSINT tasks such as sentiment analysis. The first version of the tool was delivered as a capstone project in June 2021. The social media data leveraged by the tool was provided via Sprinklr, a web scraping tool which scrapes social media applications such as Twitter and Instagram (Smithmeyer, 2021). The NPS team was also able to acquire and utilize lexicons for the Indonesian and English languages. The R-Shiny tool currently uses a lexicon-based sentiment classification methodology.

The new wartime operating environment, defined by the National Defense Strategy (NDS), makes OSINT analytic capabilities an increasingly important aspect of overall mission success in the INDOPACOM region. Adversary tactics drive the need for the U.S. to proactively engage in the social cybersecurity environment in theater. Operational units can do so by using internal assets at the Division level versus higher-echelon resources that are often dedicated to higher-priority efforts.

1.3 Research Objectives

The primary research objective of this study will be to meet or exceed the performance of state-of-the-art Indonesian emotion classification using exclusively open-source data and models. This objective will be met by examining the efficacy of

various recurrent neural network (RNN) variants within a deep neural network architecture in comparison to traditional machine learning techniques. These models will be trained and tested on a data set of labeled Indonesian Tweets for emotion classification made by Saputri et al. (2018). The specific variants examined will be Long Short Term Memory (LSTM), Bi-directional LSTM (Bi-LSTM), Stacked Bi-LSTM and Gated Recurrent Unit (GRU). These models will leverage pre-trained Word2Vec and FastText word embeddings provided by Saputri et al. (2018). Because RNNs train solely on a sequential representation of text, they have no dependence on proprietary dense features. This is in contrast to Saputri et al.'s final ensemble model which trained Logistic Regression using features such as an Indonesian Sentiment Lexicon, Bag of Words, part of speech tagging, and emoticon lists. The validity of the models will be judged based on macro-level F1 score in a 10-fold cross validation experiment.

This project can potentially be used as a methodology for performing emotion classification for OSINT cells at operational units within the INDOPACOM theater. This capability could allow OSINT cells to search for social media data based on predicted emotion class instead of simple positive or negative sentiment. For example, a user could search for all social media posts predicted to be "angry" or "fearful" for the purpose of threat identification. Emotion classification can provide characterization and understanding of Indonesian human, social, cultural and political behavior.

While this specific research is scoped by the INDOPACOM theater, it is important to note the possibility of hostile nations outside theater performing information maneuvers within its bounds. This research could produce insights by examining anomalies within the context of country of origin (i.e. an "angry" Tweet in Indonesian about the U.S. sent from some other country).

This study's use of machine learning to derive narratives from social media aligns with the NDS prioritization of technologies that "ensure we will be able to fight and win the wars of the future" (Department of Defense, 2018).

1.4 Organization

This document is organized as follows. Chapter II provides an overview of relevant background information and literature review. Chapter III details the methodology behind the various RNN models. Chapter IV presents the results of the various RNN models. Finally, Chapter V discusses the conclusions drawn from the results.

II. Literature Review

2.1 Information Warfare

In 2013, Chief of the Russian Federation's General Staff, General Valery Gerasimov published an article in the Russian Federation's Military Industrial Courier, which is considered the "genesis of hybrid or gray warfare" by the West (Bartles, 2016; Beskow and Carley, 2019). Gerasimov articulates the view that war is "conducted by a roughly 4:1 ratio of non-military and military measures. These nonmilitary measures include economic sanctions, disruption of diplomatic ties, and political and diplomatic pressure" (Bartles, 2016). These non-military measures are to be viewed as means of war, not as ways to avoid war (Bartles, 2016).

In 2015, Ben Nimmo, an analyst at Central European Policy Institute, introduced what he called *Tactics of Rebuttal* to describe the Russian Federation's social cyber-security effort to defend itself. The four tactics are dismiss, distort, distract and dismay (Nimmo, 2015). These observed tactics formed the basis for Beskow's "information maneuvers" (Beskow and Carley, 2019). The strategy is to employ a "network of officials, journalists, sympathetic commentators, and internet trolls to create an alternative reality in which all truth is relative, and no information can be trusted" (Nimmo, 2015). This approach was applied to great effect by the Russian Federation in its 2014 campaign in Ukraine (Nimmo, 2015).

In 2018, Fabio Rugge, Head of the Italian Institute for International Political Studies Centre on Cybersecurity, said the following:

> Cyberspace is a powerful multiplier of the destabilizing effects of manipulated information because it allows high connectivity, low latency, low cost of entry, multiple distribution points without intermediaries, and a total disregard for physical distance or national borders. Most importantly, anonymity and the lack of certain attribution of an attack make cyberspace the domain of ambiguity (Rugge, 2018).

In 2018, *Social Cyber-Security* was defined as "an emerging scientific area focused on the science to characterize, understand, and forecast cyber-mediated changes in human behavior, social, culture and political outcomes" (Carley et al., 2018). *Social Cyber-Security* is further defined as a "multi-disciplinary, multi-methodological, multi-level computational social science" with relevant sub-fields such as language technologies (i.e. natural language processing (NLP) and sentiment analysis), data-mining, statistics, and machine learning (Carley et al., 2018).

In 2019, Beskow and Carley defined the term *information maneuver* as the "manipulation of information and the flow or relevance of information in cyberspace" (Beskow and Carley, 2019). They list several examples of information maneuvers:

- Misdirection: introducing unrelated divisive topics into a thread in order to shift the conversation
- Hashtag latching: tying content and narratives to unrelated trending topics and hashtags
- Smoke screening: spreading content (both semantically and geographically) that masks other operations

- Thread jacking: aggressively disrupting or co-opting a productive online conversation (Beskow and Carley, 2019)

The current NDS released in 2018 acknowledged that the US security environment is "affected by rapid technological advancements and the changing character of war" (Department of Defense, 2018). It goes on to add the following:

> The fact that many technological developments will come from the commercial sector means that state competitors and non-state actors will also have access to them, a fact that risks eroding the conventional overmatch to which our Nation has grown accustomed" (Department of Defense, 2018).

The commercial sector's technological developments include advanced computing, "big data" analytics, and artificial intelligence (Department of Defense, 2018).

2.2 Sentiment Analysis

The most basic implementation of NLP is typically in the form of buzzword frequency charts or word clouds. An example of this would be an analyst producing a table of the most frequently used words that appear on Twitter from a certain area of interest. This can be a simple, yet effective method to gauge public interest. However, strict reliance on frequency-based analysis methods may "paint conclusions with broad brushes, and ignore the intricacies of language which make it so difficult to study with definitive conclusions" (Jipa, 2019; Haines, 2021). Often simple word frequency rankings will not offer the insight needed to monitor potential information maneuvers. Instead one may wish to identify broad public sentiment or emotions.

Sentiment analysis has been defined as:

> The way toward extricating data from a given dataset. It joins Natural Language Processing with Artificial Intelligence and analysis of a text. The sentiment analysis approach is utilized to mine data and deciphers individuals' feelings, against a specific topic, about any function and others. (Raisa et al., 2021)

Sentiment analysis is a sub-field of the broader discipline of NLP. "There are multiple methods for measuring sentiments, including lexical-based and supervised machine learning methods... it is unclear which one is better for identifying the polarity (i.e., positive or negative) of a message" (Ribeiro et al., 2015). Context is key to determining the sentiment of a word or phrase. "A single keyword can be used to convey three different opinions, positive, neutral and negative respectively... For example, 'fighting' and 'disease' [are] negative in a war context but positive in a medical one" (Jagtap and Pawar, 2013).

2.3 Emotion Classification

Emotion classification seeks to take the sentiment analysis approach and classify text into various human emotions instead of a binary response such as positive or negative. Emotion classification can often be more useful for general purpose sentiment mining due to the unstructured nature of social media (Saputri et al., 2018). Binary sentiment analysis is frequently more suitable for specific data sets such as movie reviews where there is a known subject and somewhat predictable format. There is no known subject or predictable format in the case of Twitter or Instagram.

A 2018 study by Saputri et al. produced the first publicly available labeled Indonesian Twitter data set for emotion classification. The study performed emotion classification on Indonesian Twitter Data using traditional Machine Learning techniques such as Support Vector Machine (SVM), Logistic Regression, and Random Forest with an ensemble of lexicons and embedding techniques (Saputri et al., 2018). The results of the study's models are published but the models themselves are not. The study also published pre-trained Word2Vec and FastText embeddings that were trained on over one million Indonesian Tweets. This study represents the state-of-the-art in emotion classification performance for Indonesian Tweets.

Traditional machine learning techniques like SVM, Logistic Regression, and Random Forest rely on "external resources or manual annotation to learn semantic and syntax features" (Zhou et al., 2018). Saputri et al.'s final model was trained on an ensemble of dense features, some of which are not publicly available (i.e. an emoticon list, an emotion word list and Vania's sentiment lexicon). Deep learning techniques like neural networks can automatically "extract the features of context" and thus have "universal applicability" (Zhou et al., 2018). The consequences of this distinction are critical to the success of this research. It is more desirable for the military to rely upon open-source data and models versus proprietary or academic ones. It is therefore the goal of this research to meet or exceed the capabilities of traditional ML techniques that leverage private data using deep learning techniques with exclusively open-source data/models.

2.4 Recurrent Neural Network Variants

LSTMs were first proposed in 1997 as a solution to RNN's issues with losing memory of its initial inputs (Hochreiter and Schmidhuber, 1997). For instance, if performing sentiment analysis on a lengthy movie review, an RNN will gradually forget the first few words (i.e. "I loved this movie, but. . . "), causing the model to lose full context of the review and potentially make an incorrect sentiment classification (Géron, 2019). LSTMs mitigate this issue by storing important inputs in long-term memory.

LSTMs with pre-trained unsupervised word vectors have been shown to be perform competitively in comparison to traditional machine learning techniques at the task of classifying movie reviews (Hassan and Mahmood, 2017). The LSTM with pre-trained word vector method can "likely overcome some flaws of bag-of-words and n-gram models" (Hassan and Mahmood, 2017). This study will directly compare the performance of an LSTM with pre-trained word vectors (i.e. Word2Vec, FastText) against traditional machine learning methods leveraging bag-of-words vectorization.

LSTMs with pre-trained unsupervised word vectors have also performed sentiment analysis on Turkish online shopping reviews and movie reviews (Ciftci and Apaydin, 2018). The LSTM model with word embeddings showed better validation accuracy in comparison to Naive Bayes and Logistic Regression models with term frequency-inverse document frequency (TF-IDF) vectorization (Ciftci and Apaydin, 2018). The study by Sapurti et al. did not leverage TF-IDF vectorization, however instead they used bag-of-words which is a similar, yet simplified form of vectorization.

LSTMs have been used to predict sentiment polarity of Indonesian Hotel reviews in order to study the effectiveness of various word embedding techniques (Imaduddin et al., 2019). Word2Vec, Glove and Doc2Vec were all paired with an LSTM classifier and the results showed Glove embeddings produced the highest validation accuracy

(Imaduddin et al., 2019). This study will only consider the two pre-trained word embedding techniques (Word2Vec and FastText) provided by Saputri et al.

Bi-LSTM in conjunction with embedding techniques have been shown to be more accurate than standard LSTMs at tasks such as sentiment analysis and emotion classification (Xiao and Liang, 2016; Elfaik and Nfaoui, 2020; Devi et al., 2021). The key difference between a Bi-LSTM and a standard LSTM is that the Bi-LSTM can "capture both past and future information", as opposed to only past information in standard LSTMs (Xiao and Liang, 2016). This allows the Bi-LSTM to capture stronger dependency relationships between words and phrases (Xiao and Liang, 2016). This study will include a Bi-LSTM model in its comparative analysis.

Devi et al.'s 2021 study of Twitter Emotion Classification for Disaster Management used Bi-LSTM for disaster classification and sentiment analysis (Devi et al., 2021). Their proposed model architecture in Table 1 combines a Glove Embedding layer and a Bi-LSTM layer with two subsequent dense layers (Devi et al., 2021). This structure outperforms standard LSTM and Bi-LSTM without an embedding layer in validation accuracy (Devi et al., 2021). This study will leverage Devi et al.'s proposed architecture for its comparative analysis.

Elfaik and Nfaoui used Bi-LSTM to perform eleven-dimensional emotion classification on Arabic Tweets (Elfaik and Nfaoui, 2020). The dataset consisted of 4,381 Arabic Tweets labeled with the following emotions: anger, anticipation, disgust, fear, joy, love, optimism, pessimism, sadness, surprise and trust (Elfaik and Nfaoui, 2020). The Bi-LSTM model outperformed several traditional machine learning baselines such as SVM, Support Vector Classifier and others when comparing validation accuracy (Elfaik and Nfaoui, 2020). This study will perform multi-dimensional emotion classification of Tweets using Bi-LSTM, albeit in five dimensions.

Table 1: Model Architecture (Reprinted, with permission, from Devi et al. © 2021 IEEE)

Various Layers and parameters:		
Layer Number	**Layer name**	**Parameters**
1	Glove embedding layer	Input dimension, Output dimension, Weight, input length
2	Bi-LSTM layer	Sentence length, return sequences, recurrent dropout (0.2)
3	GlobalMaxPool1D layer	
4	Batch Normalization	
5	Dropout Layer	Dropout rate (0.5)
6	Dense Layer	Sentence length, activation function("relu")
7	Dropout Layer	Dropout rate (0.5)
8	Dense Layer	Sentence length, activation function("relu")
9	Dropout Layer	Dropout rate(0.5)
10	Dense Layer	Sentence length, activation function("relu")
Optimizations used:		
Optimizer		Rmsprop
Loss		Binary_crossentropy
Metrics		Accuracy

Stacking multiple Bi-LSTM layers has been shown to improve sentiment analysis prediction accuracy in Chinese Micro-blog data in comparison with single layer Bi-LSTM, standard LSTM and traditional machine learning techniques (Zhou et al., 2018). It was argued that the extra layers aided in the extraction of complex features in the Chinese language (Zhou et al., 2018). This study will include a stacked Bi-LSTM architecture in its comparative analysis.

GRU is a simplified variant of LSTM and has been shown to achieve similar performance with less training time (Liang et al., 2018). Other comparisons of LSTMs and GRUs across multiple datasets have proven to be inconclusive, claiming that the choice of one versus the other is heavily dependent on the dataset and task (Chung et al., 2014). A GRU model will be leveraged in this study as part of its comparative analysis.

2.5 Limitations

A key limitation for this study's model is that it is trained exclusively on tweets determined to have one singular emotion, leaving out tweets that display multiple emotions or no emotion. This fact will impact its operational implementation. The intended use for this model's emotion prediction capability will be to predict the emotion of posts by individual users as opposed to news organizations. News updates are most often emotionless statements of fact or opinion and unfit for this model's emotion classifier. Filtering out news updates will be key to ensuring that the model makes accurate classifications. Posts with multiple emotions are not easily identifiable and filtered en masse. This study's emotion classifier is likely to predict whichever emotion is more dominant among multiple emotions.

Another limitation of this study was its limited computational resources. Each 10-fold cross validation experiment took roughly 1 to 3 hours to run using Google

Colab's GPU hardware accelerator. This limitation led the hyperparameter tuning section of this study to be performed sequentially in order of perceived importance. A dedicated high-end GPU could potentially relax this constraint and allow for robust tuning of elements such as learning rate and dropout rate.

Cambria and White defined multiple data querying techniques within the context of sentiment analysis – the first being "Keyword Spotting" where the returned content is defined by the presence of the exact word queried. The keyword spotting approach is the "most naïve and probably also the most popular because of its accessibility and economy" (Cambria and White, 2014). One of the primary issues with this approach is its "reliance upon the presence of obvious words which are only surface features of the prose" – meaning that a text document about dogs may never mention the word 'dog' because only specific breeds of dogs are discussed (Cambria and White, 2014). The Keyword Spotting method is currently being implemented by the NPS R-Shiny application as its search criteria.

Lexical affinity "assigns to arbitrary words a probabilistic 'affinity' for a particular category... For example, 'accident' might be assigned a 75% probability of indicating a negative event" (Cambria and White, 2014). These probabilities are derived from the corpus or lexicon. One of the primary issues with the Lexical Affinity approach is that it can often misinterpret context based on negation (i.e. "I avoided an accident") (Cambria and White, 2014). This method is also biased towards the corpus or lexicon utilized, meaning it cannot be broadly applied to text outside the given source's context (Cambria and White, 2014). The NPS R-Shiny app currently implements an arithmetic summation of sentiment values per word (-1 for negative, 0 for neutral and 1 for positive) derived from an Indonesian sentiment lexicon to determine sentiment of a Tweet or Instagram post.

The data correctness of the utilized lexicon is a critical factor in producing useful sentiment identification (Indulkar and Patil, 2021). In their 2021 study of Machine Learning Algorithms for Twitter Sentiment Analysis, Indulkar and Patil find a significant decrease in model accuracy for their Random Forest Classifier when using a dataset that "was not that precise and contained many words that were not in the standard language" (Indulkar and Patil, 2021). The NPS R-Shiny app currently allows the user to upload an Indonesian sentiment lexicon of their choosing.

In a 2017 study of Indonesian Public Complaints, Lailiya et al. compare the performance of machine learning algorithms performing sentiment analysis leveraging two different lexical sources: the English-based Sentiwordnet via Google-translate and an Indonesian Sentiment Lexicon called Vania. Vania produced 65% validation accuracy compared to 47% when using the Sentiwordnet method on public complaints posted to Twitter (Lailiyah et al., 2017). The authors cite social and cultural differences as the reason for the difference in accuracy, noting the Indonesian people's use of "polite language" when expressing complaints (Lailiyah et al., 2017).

A study by Koto and Rahmaningtyas in 2017 generated a social media-specific Indonesian Lexicon called InSet which outperformed the Vania general-language Indonesian lexicon by 4% accuracy (65% vs 61%) when performing sentiment analysis on Twitter data (Koto and Rahmaningtyas, 2017). These studies reinforce the potential limitations of using a sub-optimal sentiment lexicon.

Saputri et al.'s 2018 study of Indonesian emotion classification provides pre-trained embeddings in the form of Word2Vec and FastText (Saputri et al., 2018). However, multiple studies have found Glove embeddings to be the optimal embedding technique when paired with LSTM layers (Imaduddin et al., 2019; Devi et al., 2021). The exclusion of Glove embeddings could prove to be a limitation for this study's various models and their accuracy.

2.6 Primary Sources

The primary sources used for the motivation of this project will be the G2 Analytics Kickoff, the 2018 National Defense Strategy and the article on Social Cybersecurity by Beskow and Carley. The G2 Analytics Kickoff by Smithmeyer identifies key information and defines the operational unit's problem statements and objectives. This research uses those problem statements and objectives as guidance. The connection between these problem statements and objectives and the current NDS give this research a high level of significance. The article by Beskow and Carley offers a detailed analysis of the current tactics used by adversaries to influence narratives.

One primary source used for this research's methodology will be the Saputri et al. (2018) study of Indonesian emotion classification. This study's methodology will act as a comparative analysis of the utilized algorithms and published results from Saputri et al. The other primary source for this research's methodology will be Devi et al.'s (2021) study of Location Based Twitter Emotion Classification for Disaster Management. Devi et al. propose an RNN structure that will be used as the basis for this study's model structure.

III. Methodology

Preamble

This study leverages various RNN variants within deep neural network architectures to perform multi-dimensional emotion classification on Indonesian Tweets. The methodology behind deep neural networks as well as each RNN variant will be described here.

3.1 Artificial Neural Networks

The earliest version of an Artificial Neural Network (ANN) was introduced in 1943 when neurophysiologist Warren McCulloch and mathematician Walter Pitts introduced a "simplified computational model" based on the biological neuron in animal brains (Géron, 2019). The *artificial neuron* can be thought of as an object with "one or more binary (on/off) inputs and one binary output" (Géron, 2019). A network of connected artificial neurons can be used to perform logical computations as seen in Figure 1 below.

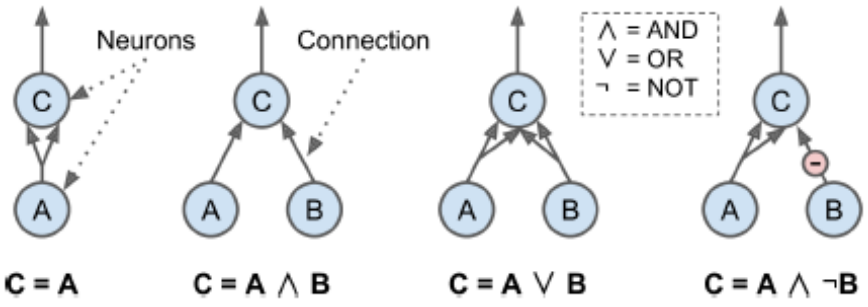

Figure 1: ANNs performing simple logical computations (Reprinted, with permission, from Géron, 2019 © O'Reilly)

In 1957, the Perceptron was introduced by Frank Rosenblatt (Géron, 2019). A Perceptron refers to a single layer of Threshold Logic Units (TLUs). TLUs are a

variant of the artificial neuron where the inputs and output consist of numbers as opposed to binary values. Additionally, each input has an associated weight assigned to it. Figure 2 below demonstrates the structure of a single TLU.

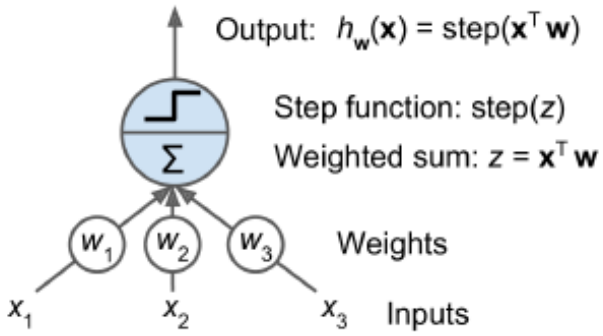

Figure 2: Threshold Logic Unit (Reprinted, with permission, from Géron, 2019 © O'Reilly)

The output of the TLU is computed by taking the weighted sum of its inputs ($z = w_1x_1 + w_2x_2 + ... + w_nx_n = x^Tw$), applying a step function to the weighted sum and outputting the result: $h_w(x) = step(z)$, where $z = x^Tw$ (Géron, 2019). A common step function is the heaviside function as seen below.

$$heaviside(z) = \begin{cases} 0, & \text{if } z < 0 \\ 1, & \text{if } z \geq 0 \end{cases} \tag{1}$$

A fully connected layer or dense layer is the result of all neurons in a layer being connected to every neuron in the previous layer. There is often an extra bias neuron, which outputs a value of 1. Figure 3 on the next page represents a Perceptron.

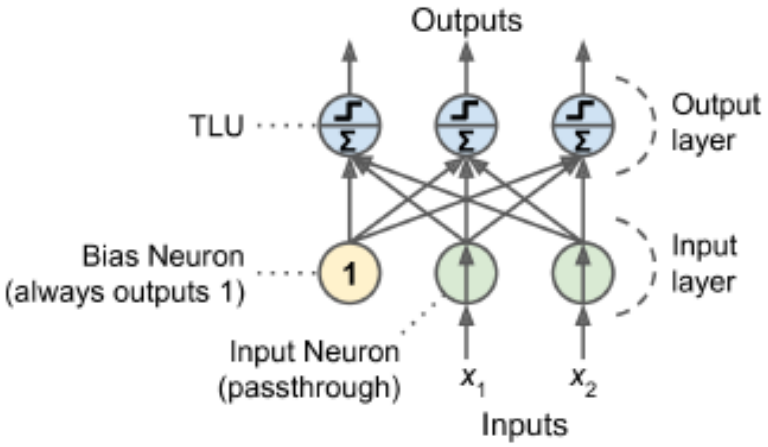

Figure 3: Perceptron Diagram (Reprinted, with permission, from Géron, 2019 © O'Reilly)

Perceptrons learn and adapt by making predictions at each training instance, then updating the weights within the network that correspond with correct predictions. For every neuron that produces an incorrect prediction, the network "reinforces the weights from the inputs that would have contributed to the correct prediction" (Géron, 2019). This iterative process is referred to as backpropogation.

A Multi-Layer Perceptron consists of an *input layer*, one or more layers of TLU's called *hidden layers* or *dense layers* and one final layer of TLU's called the *output layer* (Géron, 2019). This structure can be seen on the next page in Figure 4. When multiple hidden layers are stacked together, it is referred to as a Deep Neural Network (DNN). This structure forms the basis for each variant of Neural Network (i.e. Recurrent Neural Networks, Convolutional Neural Networks, etc.).

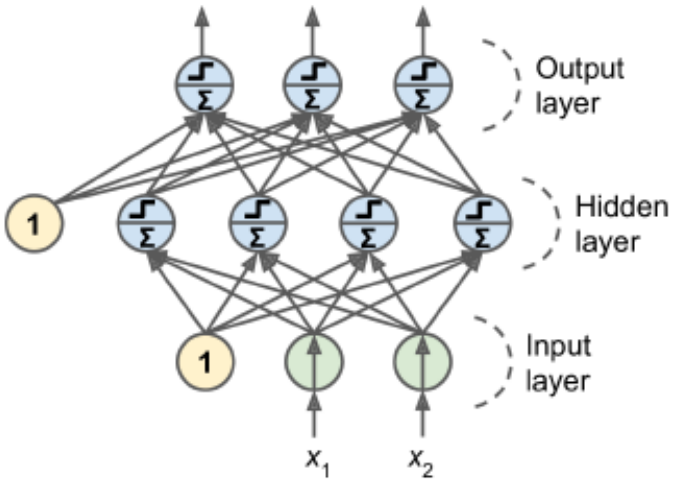

Figure 4: Multi-Layer Perceptron (Reprinted, with permission, from Géron, 2019 © O'Reilly)

3.2 Deep Neural Networks

DNNs learn by training on a portion of the training data, called a batch size, performing backpropogation and then repeating this process until all of the training data has been used. One complete pass over the entire training set is referred to as an epoch. Batch size and the number of epochs are both tunable hyperparameters. Another key tunable hyperparameter is the number of hidden layer units. This simply refers to the dimensionality of the hidden or dense layer. Increasing the number of hidden layer units can increase the DNNs ability to learn complex relationships at the cost of more computational resources.

The most popular backpropogation algorithm for training DNNs is Stochastic Gradient Descent (SGD) which calculates the derivative of an activation function. An activation function is used in place of a step function like heaviside because step functions contain only flat segments (zero derivative everywhere). Instead, activation functions like the logistic sigmoid function or hyperbolic tangent function are used, which are differentiable everywhere. SGD iterates the networks' weights in the direction of the correct response at a constant learning rate. A learning rate is a

scalar typically between values of 0.01 and 0.001. In SGD, learning rate is a critical hyperparameter for tuning (Géron, 2019).

One issue with using activation functions such as logistic sigmoid in DNNs is saturation. Saturation occurs when input values become very large in either the positive or negative direction and the resulting derivative is close to zero (see figure 5 below). This causes backpropogation to be rendered in-effective, particularly in DNNs with many layers (Géron, 2019).

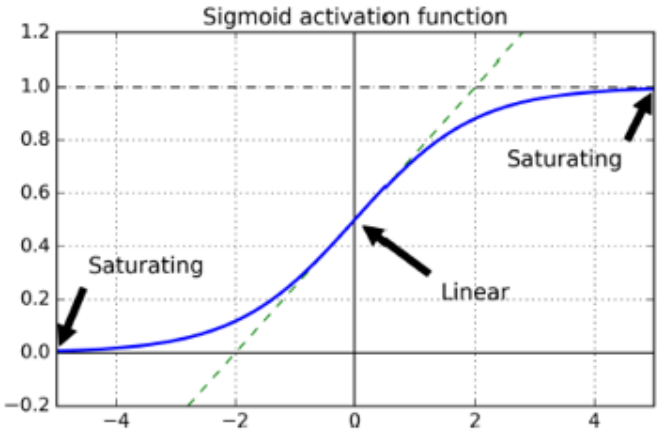

Figure 5: Logistic sigmoid activation saturation (Reprinted, with permission, from Géron, 2019 © O'Reilly)

Instead, many use a ReLU (rectified linear unit) activation function in DNN dense layers because it does not saturate for positive values (Devi et al., 2021). It is also fast to compute as the derivative of the slope is equal to one (Géron, 2019). An example ReLU activation function is illustrated on the next page in Figure 6.

Another issue that occurs in DNN is overfitting. Overfitting means that the neural network has become too biased towards the training data. Evidence of overfitting occurs when validation error increases after reaching a minima, while training error continues to decrease. Dropout has been widely used to mitigate overfitting in DNN (Devi et al., 2021). Dropout reduces overfitting by excluding input and dense layer neurons with some probability, typically between 0.2 and 0.5. Dropout effectively

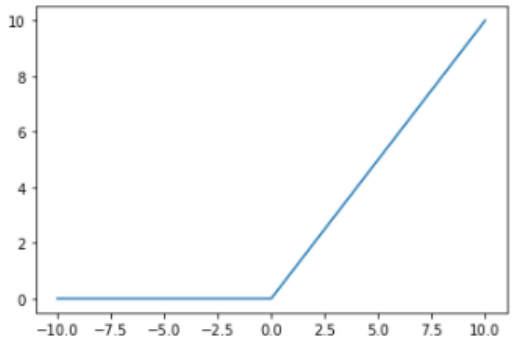

Figure 6: ReLU activation function

causes a unique neural network to be generated at each training step, resulting in an ensemble of smaller networks that are robust against overfitting. Dropout tends to slow down model convergence, however it does typically result in a significantly improved model when tuned appropriately (Géron, 2019). This process is illustrated in Figure 7 below.

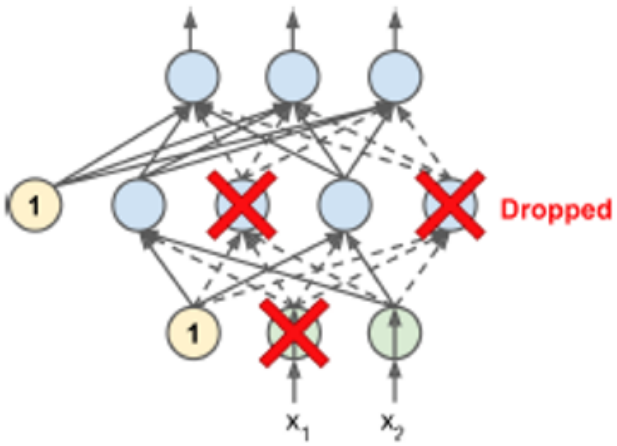

Figure 7: Dropout in a MLP (Reprinted, with permission, from Géron, 2019 © O'Reilly)

Pooling layers are another method of mitigating overfitting in DNN. The goal of a pooling layer is to shrink an input in order identify the most important feature. The pooling layer also reduces dimensionality, which helps reduce parameters and computational complexity (Géron, 2019).

RMSProp (Root Mean Squared Propogation) is a faster optimizer than SGD (Géron, 2019). It uses the concept of decay to ignore distant observations from the past and focus on more recent inputs. RMSProp is an adaptive learning algorithm and therefore "requires less tuning of the learning rate hyperparameter" (Géron, 2019). RMSProp has been used to perform emotion classification (Devi et al., 2021).

3.3 Recurrent Neural Networks

RNNs are a form of DNN that can handle sequences of arbitrary length, such as Tweets of various length. Most other DNNs are feedforward neural networks, meaning that activations only flow in one direction, from the input layer to the output layer. RNNs are structured similarly to feedforward neural networks, however they also have connections pointing backwards (Géron, 2019).

Figure 8 illustrates an RNN with a single neuron receiving inputs, producing an output and recycling that output into itself; while Figure 9 illustrates a layer of recurrent neurons.

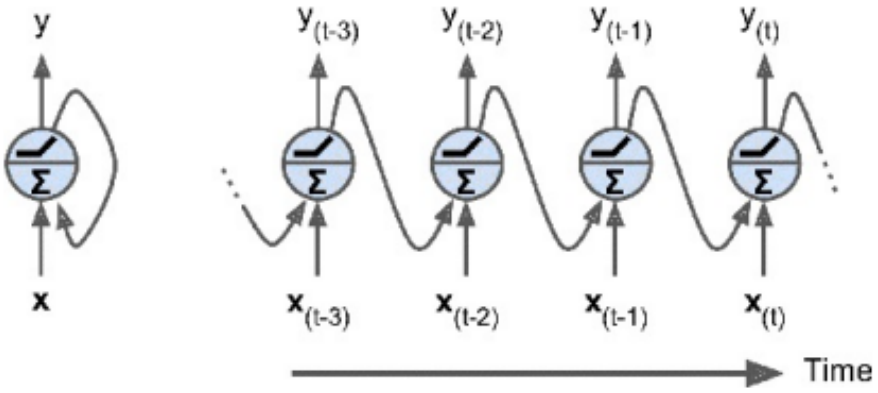

Figure 8: A recurrent neuron (left), unrolled through time (right) (Reprinted, with permission, from Géron, 2019 © O'Reilly)

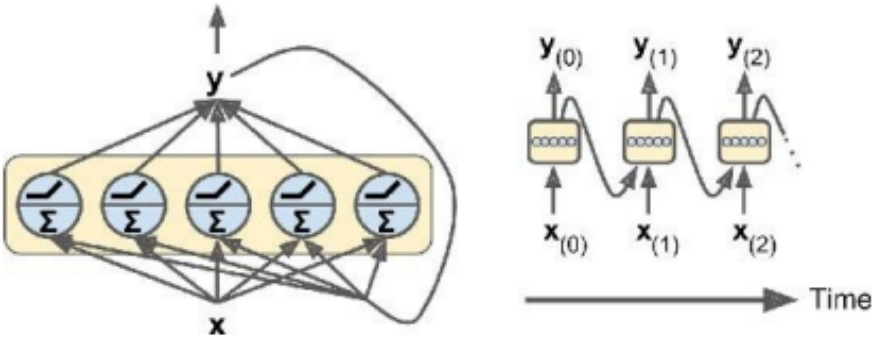

Figure 9: A layer of recurrent neurons (left), unrolled through time (right) (Reprinted, with permission, from Géron, 2019 © O'Reilly)

The recurrent neuron is a form of *memory cell* due to the fact that it "preserves some state across time steps" (Géron, 2019). A recurrent neuron is a rudimentary form of memory cell in comparison to more complex cells (i.e. LSTM or GRU). Figure 10 below demonstrates a simple recurrent neuron memory cell through time. The cell's state at any certain time step t is denoted by $\mathbf{h}_{(t)}$ and is a function of inputs from both its current and previous time steps.

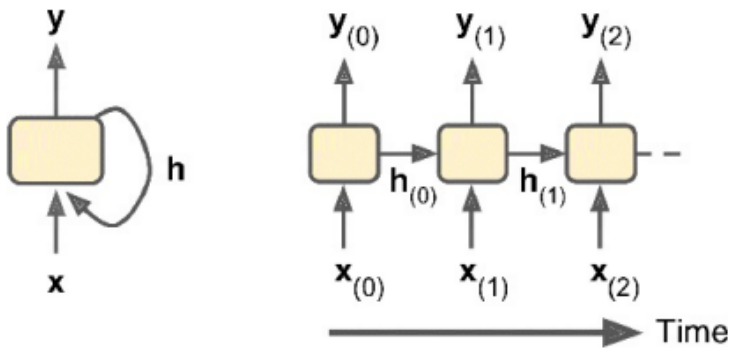

Figure 10: A simple memory cell (Reprinted, with permission, from Géron, 2019 © O'Reilly)

3.3.1 LSTM and GRU

The complexities of the LSTM cell allow for improved performance, faster training and detection of long-term dependencies in the data (Géron, 2019). Figure 11 below demonstrates the structure of the LSTM cell. The LSTM cell state stores long term information and the hidden state stores short term information. The cell and hidden states are represented by **c** and **h**. LSTM cells also possess three gate controllers that handle the storage and erasure of information from stored memory (Géron, 2019). These forget, input and output gate controllers are represented by $f_{(t)}, i_{(t)}$ and $o_{(t)}$, respectively (Géron, 2019).

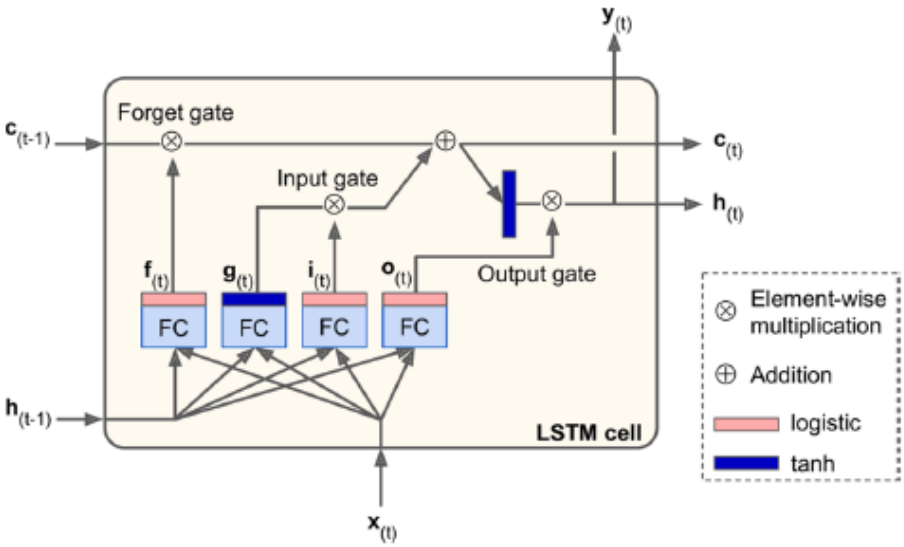

Figure 11: LSTM Cell (Reprinted, with permission, from Géron, 2019 © O'Reilly)

The GRU cell was proposed by Kyunghyun Cho et al. in 2014 and can be visualized on the next page in Figure 12 (Chung et al., 2014). The primary simplifications of GRU over LSTM are that both state vectors are merged into a single vector $\mathbf{h}_{(t)}$, a single gate controller controls the input and forget gates and the removal of an output gate (Géron, 2019).

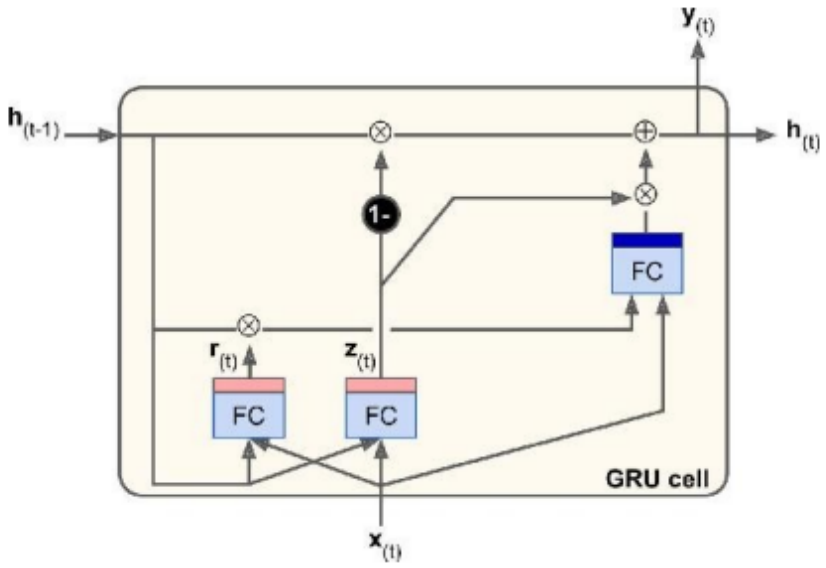

Figure 12: GRU Cell (Reprinted, with permission, from Géron, 2019 © O'Reilly)

Bi-LSTMs "consist of two LSTMs that are run in parallel: one on the input sequence and the other on the reverse of the input sequence" (Xiao and Liang, 2016). The hidden state of the Bi-LSTM is the result of the concatenation of the forward and backward hidden states at each time step, allowing the Bi-LSTM to capture both past and future information (Xiao and Liang, 2016). Figure 13 on the next page illustrates the structure of a Bi-LSTM across three time steps, highlighting the forward and backward hidden layers as well as the input and output sequences.

Similar to Bi-LSTM, Stacked Bi-LSTM can extract "rich contextual information from both past and future time sequences" (Zhou et al., 2018). However, Stacked Bi-LSTM possesses more layers to perform feature extraction, as opposed to Bi-LSTM which has only one hidden layer per direction (Zhou et al., 2018). Figure 14 on the next page illustrates the structure of a two-layer Stacked Bi-LSTM.

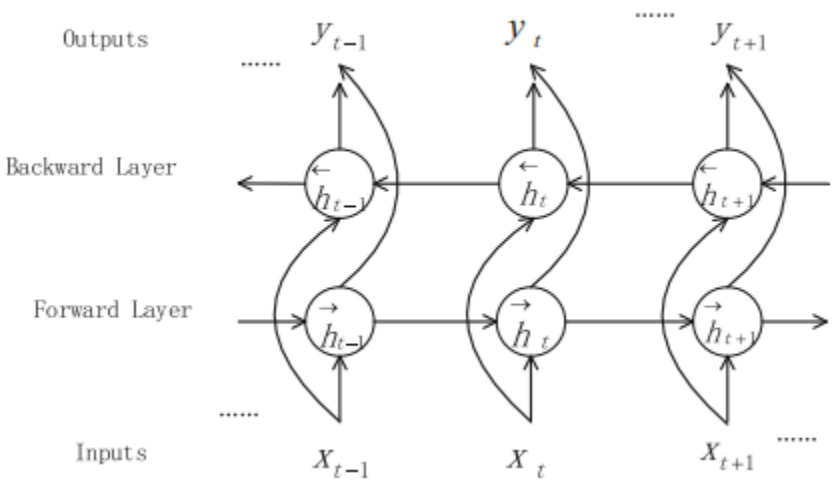

Figure 13: Bidirectional Recurrent Neural Network (Reprinted, with permission, from Xiao and Liang, 2016 © Springer)

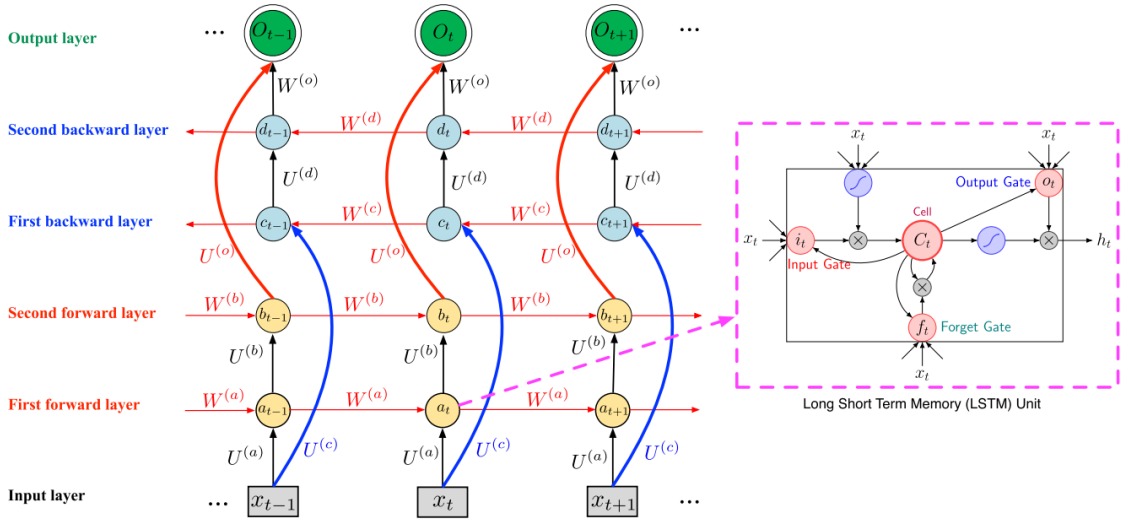

Figure 14: 2-Layer Stacked Bi-LSTM (Reprinted, with permission, from Zhou et al. © 2018 IEEE)

In Figure 14, the input sequence $\{x_1, x_2, ..., x_t\}$ enters the hidden layers in the forward direction $\{a_1, a_2, ..., a_t\}$ to extract information from all past time steps while it also passes through the hidden layers in the reverse direction $\{c_1, c_2, ..., c_t\}$ to extract information from all future time steps (Zhou et al., 2018). After this, the second hidden layers, represented by $\{b_1, b_2, ..., b_t\}$ and $\{d_1, d_2, ..., d_t\}$, receive outputs from the first hidden layers as their inputs to produce further feature extraction (Zhou et al., 2018). And finally, the output layer integrates both of the second hidden layers' output vector as its final output (Zhou et al., 2018).

Recurrent dropout has been proposed as a method of applying dropout to gated architectures such as LSTM and GRU (Semeniuta et al., 2016). Traditional dropout methods which are designed for feed-forward networks can cause loss of long-term memory when used with LSTM or GRU. Recurrent dropout is applied to the hidden state update vectors rather than the hidden states, themselves. This distinction has shown an increase in network performance using LSTMs to predict Twitter Sentiment Analysis (Semeniuta et al., 2016).

3.4 Word Embeddings

RNNs train on a sequential representation of text, making sequence-independent feature extraction processes like Bag of Words and other dense features such as emoticon lists and sentiment lexicons unsuitable for use. RNNs are, however, able to leverage word embedding techniques such as Word2Vec and FastText.

Word embeddings represent words in vector form such that the distance between vectors represents the semantic relations between respective words (Garg et al., 2018). Word2Vec is an example of a static embedding, meaning that the method learns one fixed embedding per word in the vocabulary. "The intuition of Word2Vec is that instead of counting how often each word w occurs near, say, *apricot*, we'll instead train

a classifier on a binary prediction task: 'Is word w likely to show up near *apricot*?'" (Jurafksy and Martin, 2020). This method uses self-supervision, therefore it does not require a labeled dataset. Word2Vec is "fast" and "efficient to train" on unsupervised data (Jurafksy and Martin, 2020). As such, there are many examples of pre-trained Word2Vec embeddings publicly available (Jurafksy and Martin, 2020). For example, Saputri et al. make publicly available a 400-dimension Word2Vec embeddings model that was trained on over one million Indonesian Tweets.

FastText is another static embedding technique and an extension of Word2Vec. FastText represents words as themselves along with a "bag of constituent n-grams" (Jurafksy and Martin, 2020). For instance, if $n = 3$, the word *there* would be represented by the sequence **there** along with the character n-grams: <**th, the, her, ere, re**>. Then the skipgram embedding is learned for each constituent n-gram and the word *there* is represented by the sum of all the embeddings of its constituent n-grams (Jurafksy and Martin, 2020). Saputri et al. also provide a pre-trained 100-dimensional FastText model trained on the same set of Indonesian Tweets.

3.5 Dataset

Saputri et al. produced a dataset of 4,403 Indonesian Tweets labeled into five emotion classes: love, anger, sadness, fear and joy. This dataset was labeled by two individuals and evaluated using a Cohen Kappa measurement – a statistic for measuring interrater reliability (Saputri et al., 2018). A score above 0.81 is considered to be "almost perfect agreement" between two raters (McHugh, 2012). This study produced a Kappa score of 0.917 (Saputri et al., 2018). Multi-emotion tweets and no-emotion tweets were excluded from this dataset (Saputri et al., 2018). The data classifications are slightly imbalanced with the distribution seen on the next page in Figure 15.

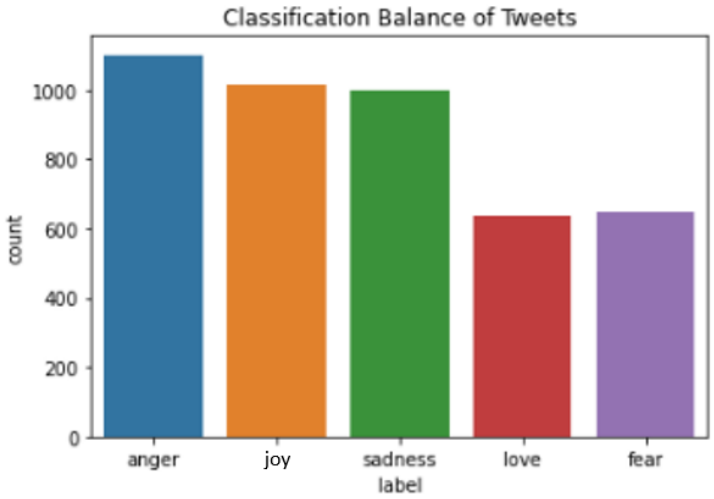

Figure 15: Classification Balance of Tweets

Saputri et al.'s data set contains emoticons (i.e. **:)** expressing joy). Saputri et al. created an unpublished list of emoticons and their respective emotion. This study will not recreate an emoticon list, but instead allow the models to learn such features automatically.

The tweets produced by the 2018 study already have some level of pre-processing. Usernames with the @ symbol have been replaced with the generic [USERNAME], URL's and hyperlinks have been replaced with the generic [URL] and sensitive numbers such as phone numbers, invoice numbers, and courier tracking numbers have been replaced with the generic [SENSITIVE-NO] (Saputri et al., 2018).

The preprocessing steps defined by Raisa et al. will be utilized by this research: eliminating deficient and conflicting information, URLs, punctuation, numbers, other special characters, stop word removal, and tokenization (Raisa et al., 2021). Stop words are defined as extremely common words which are of little value such as "a, an, he, is, it" (Manning, 2008). Tokenization refers to splitting a sentence into separate "tokens", often delineated by white space (Manning, 2008).

The data and pre-trained embeddings were downloaded from Saputri et al.'s GitHub page in CSV form. Additional pre-processing was performed on the tweets by removal of stop words via the Natural Language Toolkit (NLTK) Indonesian Stop Word library. Tokenization is performed using Keras' Tokenizer function, which converts the tweets into sequences of integers. Each tweet is set to the same length by padding the sequences with zeroes up to the maximum length of the longest tweet.

The pre-trained embeddings are then loaded into a dictionary and cross-referenced with the words from the dataset of Tweets. This process results in finding each word in the dataset's vector representation. The embeddings are stored in a matrix the size of the maximum tweet length times the dimensionality of the pre-trained embeddings (i.e. 65x400 or 65x100). Words that exist in the tweets but not in the embeddings are given a value of zero.

3.6 Evaluation Metrics

The Saputri et al. (2018) study produced a comparative study of various machine learning techniques leveraging many different dense features – see Table 2 on the next page. This research will compare the results of its various RNN models using Word2Vec and FastText pre-trained embeddings against the Saputri study's Logistic Regression model on the "New Dataset". This research will also perform a direct comparison of its models to Saputri's using only word embedding features (i.e. Word2Vec and FastText). Saputri et al. used 10-fold cross-validation to split their data into training and test sets. The average Macro-level F1 score across all 10 folds are displayed in Table 2. This study will identify the optimal model based on identifying a global maxima for validation accuracy for each of the 10 folds and taking the average. This study will also examine precision, recall and F1 score for predictions of each respective class from the model with the best averaged macro-level F1 score.

Table 2: Comparative Macro-Level F1 Scores (Reprinted, with permission, from Saputri et al. © 2018 IEEE)

Features	The's Dataset [4]			New Dataset		
	LR	SVM	RF	LR	SVM	RF
Basic Features						
Emotion word's list (EW) [8]	57.85%	58.34%	56.04%	43.09%	43.12%	42.20%
Bag-of-Words (BOW)	69.53%	67.52%	67.31%	65.13%	57.88%	64.07%
Word2Vec (WV)	67.32%	53.96%	40.32%	61.83%	61.37%	53.03%
FastText (FT)	66.46%	65.42%	55.01%	62.49%	62.27%	55.18%
EW + BOW + WV	70.34%	64.77%	62.52%	68.25%	61.20%	59.60%
EW + BOW + FT (Basic)	73.72%	71.46%	64.53%	68.39%	61.58%	62.23%
Lexicon						
Vania's Sentimen (VSent) [16]	11.02%	11.59%	11.71%	8.15%	8.79%	9.59%
InSet Sentimen (ISent) [17]	22.34%	19.05%	28.17%	19.36%	14.78%	24.92%
Emoticon List (Emot)	11.56%	11.53%	13.05%	11.45%	11.52%	11.48%
VSent + ISent + Emot (Lex)	23.80%	19.16%	28.85%	22.48%	15.37%	25.87%
EW + Lex	62.91%	62.19%	55.78%	50.30%	37.08%	46.56%
Other Features						
Orthographic (Ort)	20.39%	16.19%	26.62%	21.16%	9.25%	25.31%
Pos Tag (POS)	16.87%	16.91%	17.37%	16.79%	17.14%	17.84%
Ort + POS	27.93%	16.93%	30.66%	22.98%	15.00%	25.36%
Features Combination						
Basic + Lex	74.60%	68.43%	65.94%	69.43%	63.06%	62.30%
Basic + Ort + POS	74.60%	69.90%	66.90%	69.22%	57.99%	62.72%
Basic + Lex + POS + Ort	**75.98%**	70.84%	66.,60%	**69.73%**	64.24%	63.06%

Precision is defined as the ratio of True Positive results over the Total Predicted Positive (True Positive + False Positive) results. Recall is defined as the ratio of True Positive over the Total Actual Positive (True Positive + False Negative). F1 score represents the harmonic mean of precision and recall and is defined by the following equation:

$$\mathbf{F1} = \mathbf{2} \times \frac{Precision \times Recall}{Precision + Recall} \tag{2}$$

Macro-F1 score is the simple mean of each of the classes F1 scores, unweighted by class size. Micro-F1 accounts for the weight of classes based on sample size. Macro-F1 scores are typically used when balanced classes are present.

3.7 Model Architecture

This experiment consists of four models: LSTM, GRU, Bi-LSTM and Stacked Bi-LSTM. Parameters were held constant across each model with the exception that hidden layer units in Bi-LSTM layer were halved to maintain dimensionality. The models are run for 20 epochs for each fold in the 10-fold cross-validation experiment. Batch size is fixed to 64 for each epoch. The RMS Prop optimizer uses the default learning rate of 0.001 and default decay rate, rho, of 0.9. Loss is measured by Categorical Crossentropy.

Table 3 illustrates the architecture of the LSTM, GRU, and Bi-LSTM Models. The first layer of the model is the embedding layer which uses the pre-trained Word2Vec or FastText embeddings from Saputri et al. This layer converts each word in the corpus of tweets to a 400 or 100-dimensional vector representation, respectively. The input length is set to the maximum length of the longest tweet in the dataset (i.e. 65 words), thus every tweet is padded with zeroes up to that length. The size of the output of the embedding layer is a 65 x 400/100 matrix for every unit in the batch (i.e. in Table 3 represented as (None, 65, 400/100) indicating that the tweets have not yet been batched). The embedding layer is trainable, meaning that the model will adjust the values of the embeddings during backpropogation. This matrix then passes through a 1-dimensional Spatial dropout layer with a rate of 0.2 to avoid overfitting.

The output of the Spatial dropout layer feeds into the LSTM/GRU/Bi-LSTM layer with 512 units (256 x 2 for the Bi-LSTM layer). Thus, the output of the LSTM/GRU/Bi-LSTM layer is a 65 x 512 matrix. The GRU layer has slightly fewer parameters than the LSTM layer due to its simplicity. The Bi-LSTM's output shape accounts for both the forward and backwards directions, therefore in order to have an output dimension of 512, the Bi-LSTM layer must possess half the cell units or in this case, 256. This accounts for the Bi-LSTM layer having less parameters than both the

LSTM and GRU layers in Table 3. The LSTM/GRU/Bi-LSTM layer has dropout at its inputs of 0.2 and recurrent dropout of its recurrent state of 0.2 to avoid overfitting (Semeniuta et al., 2016). This output is then fed into a 1-dimensional global max pooling layer. The output of the Global Max Pooling layer is a vector of length 512.

Table 3: LSTM/GRU/Bi-LSTM Models in Keras

Layer	Output Shape	Param #
(Word2Vec/FastText Embedding)	(None, 65, 400/100)	8M/2M
(Spatial Dropout 1-D)	(None, 65, 400/100)	0
(LSTM/GRU/Bi-LSTM)	(None, 65, 512)	~1.2M/0.9M/0.7M
(Global Max Pooling)	(None, 512)	0
(Dense #1)	(None, 512)	262656
(Dropout)	(None, 512)	0
(Dense #2)	(None, 256)	131328
(Dropout)	(None, 256)	0
(Dense Output)	(None, 5)	1285

Next, the output vector is fed into two dense layers of decreasing size. The first dense layer is of size 512 and the second is of size 256. Both layers utilize ReLU activation functions. Dropout is applied after both dense layers at a rate of 0.5 to avoid overfitting. The final output layer reduces the vector to length 5 with a softmax activation function which converts the vector of numbers into a vector of probabilities. Softmax returns the index of the highest probability (i.e. the predicted emotion).

The stacked Bi-LSTM model can be seen below in Table 4. The 2 layers of Bi-LSTMs are stacked upon one another. The dimensionality of the other models is preserved; however, the number of parameters is increased from roughly 3 million to 4.7 million. This increase in parameters will likely increase the training time of the stacked Bi-LSTM model. This distinction will be monitored during model evaluation.

Table 4: Stacked Bi-LSTM Model in Keras

Layer	Output Shape	Param #
(Embedding)	(None, 65, 100)	2000000
(Spatial Dropout 1-D)	(None, 65, 100)	0
(Bi-LSTM)	(None, 65, 512)	731136
(Bi-LSTM)	(None, 65, 512)	1574912
(Global Max Pooling 1-D)	(None, 512)	0
(Dense #1)	(None, 512)	262656
(Dropout)	(None, 512)	0
(Dense #2)	(None, 256)	131328
(Dropout)	(None, 256)	0
(Dense Output)	(None, 5)	1285

Total params: 4,701,317
Trainable params: 4,701,317
Non-trainable params: 0

IV. Results and Analysis

4.1 Programming Platform

This work was performed using the Python Programming language (3.7.12) with the following key packages: Numpy (1.19.15), Pandas (1.1.5), Keras (2.5.0), SkLearn (0.22) and NLTK (3.4.5). See Appendix A for full Model Code. The primary experiments were conducted in a Google Colab environment using Google Colab's GPU hardware accelerator. Google Colab utilizes GPUs with varying performance depending on the particular connection instance, therefore compute times cannot be reliably tracked for comparative purposes. Additional time trials were conducted using an AMD Ryzen 5 2600 6 Core, 12 Thread Processor.

4.2 Cross Validation Results

The results of this study's cross validation comparative experiment can be seen below in Tables 5 and 6. In Table 5, this research's models are compared to Saputri's models using only embedding techniques. In Table 6, this research's models are compared to the best model from Saputri et al. that uses various dense features. These values represent the mean of the maximum validation accuracies for each of the 10 K-Fold splits.

Table 5: Embedding Technique Comparison (Saputri et al., 2018)

	Glenn 2022				Saputri et al. 2018		
Embedding	LSTM	GRU	Bi-LSTM	Stacked Bi-LSTM	LR	SVM	RF
FastText	70.48%	70.17%	**70.71%**	**70.71%**	62.49%	62.27%	55.18%
Word2Vec	65.92%	64.90%	66.33%	65.01%	61.83%	61.37%	53.03%

Each of the models produced by this research outperform each model from Saputri et al. when embedding techniques are the only features considered. The best models

from this study (FastText Bi-LSTM and 2-layer Stacked Bi-LSTM) outperform Saputri et al.'s Logistic Regression model with FastText embeddings by more than an 8% margin of accuracy in 10-fold cross validation. The best models from this research also outperform Saputri et al.'s Random Forest model by more than 15%.

Table 6: Comparison Using All Features (Saputri et al., 2018)

	Glenn 2022				Saputri et al. 2018
Embedding	**LSTM**	**GRU**	**Bi-LSTM**	**Stacked Bi-LSTM**	**LR + features**
FastText	70.48%	70.17%	**70.71%**	**70.71%**	69.73%
Word2Vec	65.92%	64.90%	66.33%	65.01%	----

Each of the models with FastText embeddings outperform the best model from Saputri et al. when all features are considered, while each of the models using Word2Vec embeddings under-perform relative to their FastText counterpart. The increase in performance when using FastText over Word2Vec is consistent with the results of the Logistic Regression model from Saputri et al. (seen in Table 2 on page 31). The best performing models are the FastText Bi-LSTM and FastText 2-layer Stacked Bi-LSTM models with the same validation accuracy of 70.71% (seen emboldened in Table 6). The Stacked Bi-LSTM model has significantly more parameters than the single layer Bi-LSTM and yet, does not produce higher validation accuracy. The distribution of results over 10 folds for the FastText RNN variants can be seen in Figure 16.

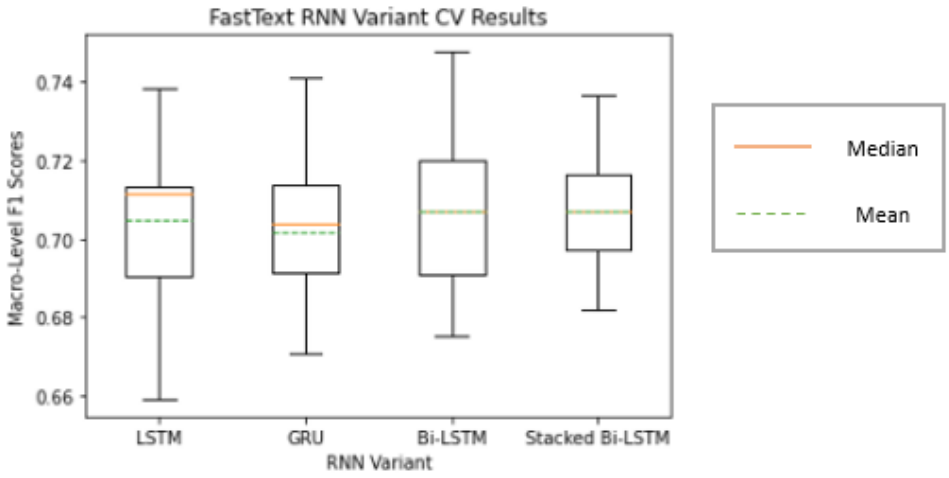

Figure 16: FastText RNN Variant Cross Validation Results

4.3 Time Trials

Time trials were conducted on a local machine in order to further examine the performance of these models. These models are identical to the FastText models from the Cross Validation experiment in the previous section. The results represent the compute time of a 90/10 training and validation split of the data trained for 20 epochs. The resultant compute times can be seen below in Table 7.

Table 7: FastText Time Trials

	Glenn 2022			
Embedding	**LSTM**	**GRU**	**Bi-LSTM**	**Stacked Bi-LSTM**
FastText	0:50:40	0:35:24	**0:24:03**	2:18:20

* Duration format: h:mm:ss

The single layer Bi-LSTM model can achieve the same accuracy as the 2-layer stacked Bi-LSTM in less than one-fifth of the compute time. This suggests that the single layer Bi-LSTM model with FastText embedding may be optimal, although this cannot be verified without a test for statistical significance.

4.4 Statistical Significance

A statistical test such as ANOVA cannot be used to compare multiple classifiers based on the means of a cross validation experiment because cross validation violates the assumption of statistical independence (Raschka, 2018). Therefore it cannot be stated that any of the models in this study are statistically better than the final model from Saputri et al. However, an internal comparison can be performed to determine any statistical significance among this study's models using the bootstrapping method. Bootstrapping is similar to cross validation but with sample replacement. This distinction satisfies the ANOVA test's assumption of statistical independence among samples. The results of the bootstrapping experiment with 10 training splits can be seen below in Table 8 and Figure 17.

Table 8: 10-Sample Bootstrap Macro-Level F1 Scores

	Glenn 2022			
Embedding	**LSTM**	**GRU**	**Bi-LSTM**	**Stacked Bi-LSTM**
FastText	68.26%	68.56%	**68.62%**	68.09%

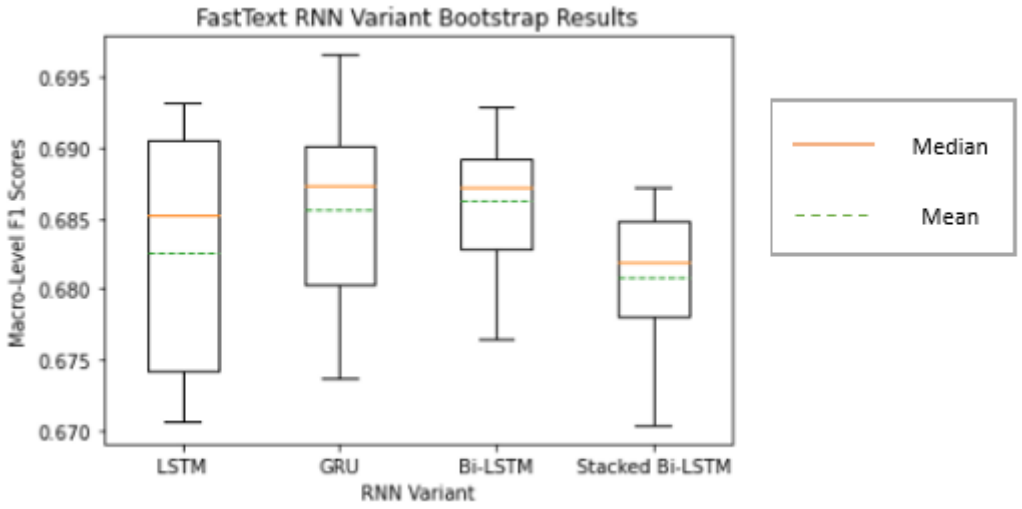

Figure 17: FastText RNN Variant Bootstrap Results

A single-factor ANOVA was performed on the bootstrap comparative results with a null hypothesis that all classifier means are equal and an alpha of 0.05. The single-factor ANOVA fails to reject the null hypothesis (F-stat = 1.35 and p-value = 0.27), meaning that the data cannot conclude that the four classifiers' means are not equal. In addition, post hoc pairwise t-tests were conducted with a null hypothesis that the mean difference between two classifiers is zero and an alpha of 0.05. For the comparison of the single layer Bi-LSTM model and the 2-layer stacked Bi-LSTM model, this test rejects the null hypothesis (p-value = 0.037) which reveals a statistical significance between the two models. All other pairwise t-tests were found to be statistically insignificant. Therefore, this research can conclude that the single layer Bi-LSTM model is optimal in comparison to the 2-layer stacked Bi-LSTM model.

4.5 Hyperparameter Tuning

The single layer Bi-LSTM model with FastText embedding was chosen as the candidate model for hyperparameter tuning based on its statistical significance over the 2-layer stacked Bi-LSTM, and it possessing the highest macro-level F1 score in both the cross validation and bootstrap comparative experiments, as well as the fastest compute time. Hyperparameter tuning can identify model dependencies that can potentially be exploited in order to boost performance. The first hyperparameter for consideration is the number of Bi-LSTM cell units. Variation of the number of units in the Bi-LSTM layer produces the following results in Table 9. Increasing the number of units in the Bi-LSTM layer from 256 to 512 leads to an increase in Macro-Level F1 score up to 70.83%. However, increasing the cell units up to 1024 shows a decrease in performance.

Table 9: Cell Unit Tuning Macro-Level F1 Scores (Saputri et al., 2018)

Glenn 2022 Single-Layer Bi-LSTM				Saputri et al. 2018 LR
Cell Units	256	512	1024	N/A
Val Accuracy	70.71%	**70.83%**	70.64%	69.73%

Lowering the batch size can increase accuracy at the cost of additional run time. Variation of batch size with Bi-LSTM layer set to 512 can be seen below in Table 10. Decreasing the batch size below 64 did not produce higher accuracy. Notably, increasing the batch size to 128 also caused a decrease in validation accuracy, suggesting 64 is an optimal batch size.

Table 10: Batch Size Tuning Macro-Level F1 Scores (Saputri et al., 2018)

Glenn 2022 Single-Layer Bi-LSTM 512 Units					Saputri et al. 2018 LR
Batch Size	16	32	64	128	N/A
Val Accuracy	69.44%	70.28%	**70.83%**	70.42%	69.73%

After tuning, the best model is determined to be a FastText Single-Layer Bi-LSTM with 512 units in the Bi-LSTM layer and a batch size of 64. This model is able to produce a study-best 70.83% accuracy average across 10-fold cross validation. The final model architecture can be seen on the next page in Table 11. Note: the output shape of the Bi-LSTM layer is twice the Bi-LSTM layer units to account for both the forward and backwards directions (i.e. 512 x 2 = 1024).

Tunable elements such as learning rate, dropout rates and the number of neurons in dense layers 1 and 2 were not tuned due to computational resource constraints. Learning and dropout rates largely impact convergence time within a certain number of epochs. The model was found to converge to a global maximum validation accuracy within the given number of epochs during every training split during the cross

Table 11: Final Single-Layer Bi-LSTM Model in Keras

Layer (type)	Output Shape	Param #
(Embedding)	(None, 65, 100)	2000000
(Spatial Dropout 1-D)	(None, 65, 100)	0
(Bi-LSTM)	(None, 65, 1024)	2510848
(Global Max Pooling 1-D)	(None, 1024)	0
(Dense #1)	(None, 512)	524800
(Dropout)	(None, 512)	0
(Dense #2)	(None, 256)	131328
(Dropout)	(None, 256)	0
(Dense Output)	(None, 5)	1285

Total params: 5,168,261
Trainable params: 5,168,261
Non-trainable params: 0

validation experiment. Additionally, the RMSprop optimizer is an adaptive learning algorithm which makes the default learning rate suitable in most cases. The number of neurons in the dense layers were chosen to preserve the dimensionality of the RNN layer's output with a stepwise decrease until the final dense layer.

4.6 Model Examination

The Single-Layer Bi-LSTM Model will now be further examined to show its behavior during training. The model's accuracy and loss values over 20 epochs can be seen below in Figures 18 and 19. These values were extracted from one of the K-Fold Cross Validation training splits.

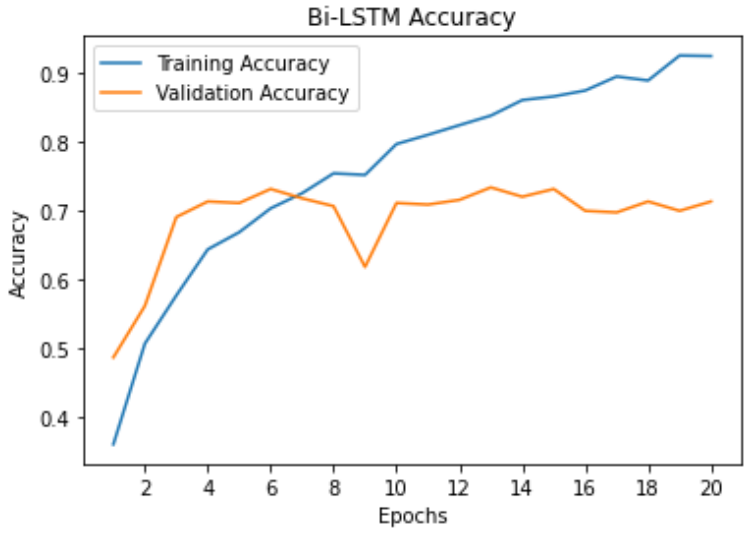

Figure 18: Single-Layer Bi-LSTM Model Accuracy

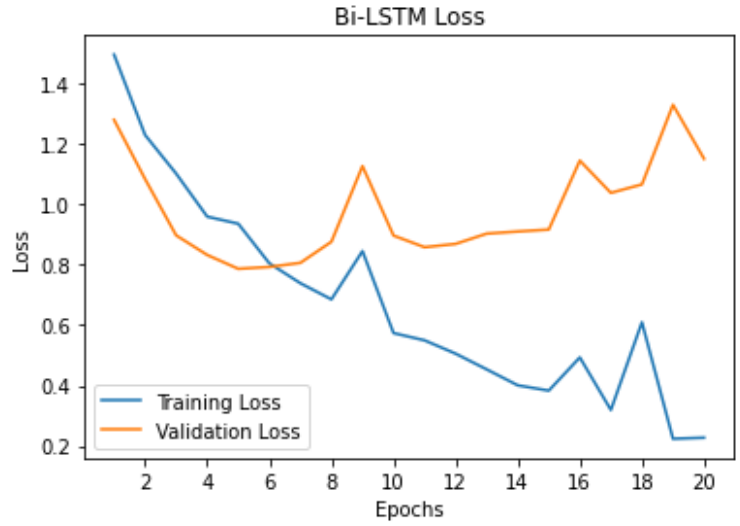

Figure 19: Single-Layer Bi-LSTM Model Loss

The optimal model was identified by maximizing the validation accuracy at epoch 13 (73.41%). The slight increase in validation loss after epoch 5 suggests some overfitting, however this increase is not significant enough to rule out the optimal model at epoch 13. It is noteworthy that validation accuracy at epoch 5, the validation loss minimum, is 71.14%, which is still greater than the optimal model from Saputri et al. This suggests that the model is robust against overfitting.

Examination of the model's classification report reveals that the fear class has the highest level of precision at 84%, which indicates a false positive rate of 16%. Recall for the fear class is also high with a score of 71%. This performance is in contrast with the baseline model from Saputri et al., which performs relatively poorly in the fear class with a precision rate of 65% and recall rate of 53%. The classification reports from this study and Saputri et al. can be seen below in Tables 12 and 13.

Table 12: Classification Report (Glenn, 2022)

Class	Precision	Recall	F1-Score
love	78%	88%	83%
joy	77%	65%	71%
anger	75%	85%	80%
sadness	62%	65%	64%
fear	84%	71%	77%
avg/total	**76%**	**75%**	**75%**

Table 13: Classification Report (Reprinted, with permission, from Saputri et al. © 2018 IEEE)

Class	Precision	Recall	F1-Score
love	64%	75%	69%
joy	81%	60%	69%
anger	61%	81%	70%
sadness	89%	72%	80%
fear	65%	53%	59%
avg/total	**70%**	**68%**	**68%**

Examination of the fear and anger classes in Table 12 reveals diverging successes when evaluating the classes based on precision and recall metrics. Precision is a metric often used when the cost of a false positive is high while recall is often used when the cost of a false negative is high. In a military intelligence context, failing to identify a potential threat (i.e. an angry or fearful tweet) could prove costly. If this model were to be provided to a social media threat analyst, perhaps the optimal criteria would be to maximize recall in order to provide the user with more data rather than less in the interest of reducing false negatives.

The confusion matrix below in Figure 20 demonstrates the classification performance of the model in greater detail. It can clearly be seen that the model produces the most true positives in the fear category - meaning 84% of its "fear" predictions are actual "fear" tweets according to the labeled data set.

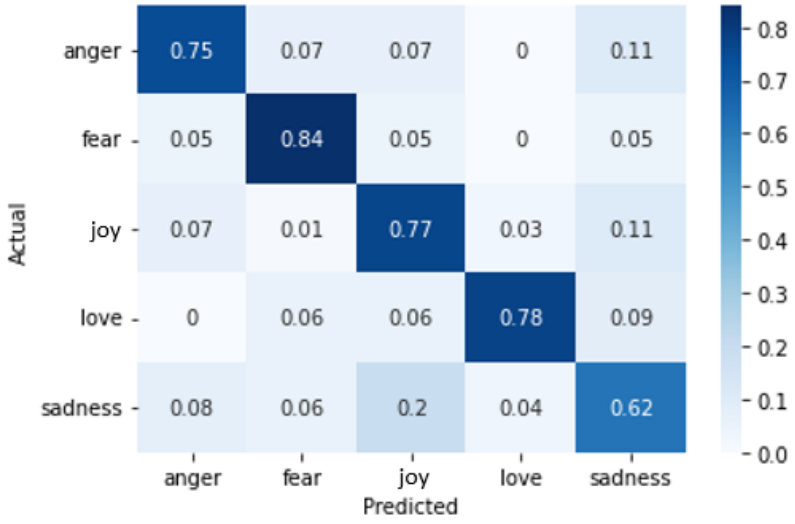

Figure 20: Single-Layer Bi-LSTM Confusion Matrix

The model's worst performing category is "sadness", which could be due to sarcasm. Indonesians frequently use sarcasm and polite language when making complaints (Lailiyah et al., 2017). This could explain why sadness is mistaken for joy 20% of the time. An example of this can be seen using the following tweet. The tweet

in Indonesian states "pengen ikut indonesia idol tapi aku g punya bakat nyanyi :D" which translates to "I want to join Indonesian idol but I don't have singing talent :D" according to Google Translate. The Single-Layer Bi-LSTM model predicts that this tweet's emotion is joy, likely due to the emoticon ':D' indicating an open mouth smile. However, this tweet could be intended as sarcasm and classified as 'sadness' instead. This ambiguity highlights the difficulty in determining the emotion of social media posts even for a human.

The "sadness" emotion class is the only class where the Saputri et al. model outperforms this study's model in terms of F1 score (see tables 11 and 12 on page 42). The ability to predict negative emotions such as sadness, fear and anger is more important than positive emotions such as joy or love in a military intelligence context. And in terms of the importance of negative emotions relative to one another, fear and anger are more noteworthy for a social media threat analyst than sadness. For instance, a tweet such as "Polisi Indonesia Biadab" which translates to "Barbaric Indonesian Police" is more consequential than a tweet lamenting one's inability to make it onto Indonesian idol (see tweet from previous paragraph). A further examination of each RNN variants' performance by response class can be seen in Appendix B.

The model does not consider the possibility of emotionless or neutral posts as mentioned in the limitations section. This limitation could require the user to filter out posts suspected to be neutral such as news posts, although this workaround will likely not be sufficient as there will certainly exist many posts from non-news sources that exhibit neutral emotion. In order to demonstrate the importance of this limitation, a collection of 1125 non-news Indonesian Tweets predicted to be "neutral" by Sprinklr's sentiment analysis classifier were given to the model for emotion classification. The results below in Table 14 show that "neutral" posts are predicted to be either "anger" or "joy" more than two-thirds of the time. The ability to properly han-

dle "neutral" emotions will be critical to avoid over-classifying the "anger" emotion and therefore diminishing the model's ability to provide an accurate representation of public behavior.

Table 14: Neutral Post Classification

Emotion Prediction of "Neutral" Tweets				
Anger	**Fear**	**Joy**	**Love**	**Sadness**
33%	6%	39%	9%	13%

* 1125 Indonesian Tweets classified as "Neutral" by Sprinklr

V. Conclusions

The results of this study demonstrate the efficacy of RNNs when leveraging pre-trained FastText embeddings in comparison to traditional machine learning techniques. Specifically, these results show that RNN variants can produce more than an 8% gain in accuracy in comparison to Logistic Regression and SVM techniques and a 15% gain over Random Forest. This research found a statistical significance in the performance of a single layer Bi-LSTM model over a 2-layer stacked Bi-LSTM model. This research also found that single layer Bi-LSTM models produce comparable macro-level F1 when compared to the best model from Saputri et al. The final ensemble method from Saputri et al. was a logistic regression model leveraging FastText pre-trained embeddings, bag-of-words feature extraction, an emotion word list, and several other lexical features for a final macro-level F1 score of 69.73%. This study's single layer Bi-LSTM model achieves a macro-level F1 score of 70.83% using only the pre-trained FastText embeddings. This suggests that RNNs are successfully able to automatically extract the dense features manually provided in the legacy study (i.e. emoticons, parts of speech, etc.). These results satisfy this research's goal of meeting or exceeding the performance of Saputri's legacy machine learning methods using exclusively open-source data and models.

This single layer Bi-LSTM emotion classification model can be provided to operational units within INDOPACOM in order to enhance their OSINT capabilities. The emotion classification model can provide characterization and understanding of Indonesian behavior through analysis of its social media data. OSINT cells can use this new capability to search for social media posts based on prediction emotion class in order to gauge public reaction to military actions in theater. These capabilities align with the NDS's prioritization of technological developments in artificial intelligence that can change the character of war.

5.1 Future Work

Future work could include producing a model that handles tweets with neutral emotion or multi-emotion tweets. A model that could handle neutral emotion posts would be particularly useful for operational implementation as it would remove the need for filtering posts before making predictions. Saputri et al. indicate they may produce a dataset in the future that has multi-emotion and neutral-emotion labeling (Saputri et al., 2018). It may be necessary to acquire additional labeled data from other Indonesian sources in order to train a model that can predict neutral and multiple emotions. A potential solution could be to create a customized labeled dataset of Indonesian tweets with the assistance of language experts.

Other research areas such as topic modeling and network analysis could be applied in conjunction with this emotion classifier model in order to further expand upon its ability to provide characterization and understanding of societal behaviors. For example, network analysis could delineate social groups by examining Twitter interactions and emotion classification could be used to determine how those social groups respond to specific topics.

Appendix A. Model Code

```
## source: https://towardsdatascience.com/word-embeddings-for-sentiment-analysis
-65f42ea5d26e
## source: https://towardsdatascience.com/twitter-topic-modeling-e0e3315b12e2
```

```
import pandas as pd
import numpy as np
import re
import collections
import matplotlib.pyplot as plt
import csv
import seaborn as sns

import nltk
from nltk.corpus import stopwords

import keras
from keras import layers, regularizers, Sequential
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils.np_utils import to_categorical
from keras.layers import *
from keras.models import Model
from keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import RMSprop
from keras.losses import CategoricalCrossentropy
from keras.metrics import Accuracy

import sklearn
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, KFold
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.utils import resample
from sklearn.metrics import multilabel_confusion_matrix, classification_report,
confusion_matrix
```

```
def custom_tokenizer(df, tk):
    """
    Tokenizes tweets using Keras' Tokenizer after stop word removal

    Args:     df, Pandas DataFrame: Contains all tweets and labels
              tk, Keras Tokenizer Object:  Creates tokens via fit_on_texts fn

    Returns:  df, Pandas DataFrame:  Updated DataFrame with tokenized tweets in
 new column
    """

    tokenizeList = []

    # remove stopwords
    df.tweet = df.tweet.apply(remove_stopwords)

    tweetList = df['tweet'].tolist()
    tk.fit_on_texts(df['tweet'])
    inv_map = {v: k for k, v in tk.word_index.items()}

    for sentence in tweetList:
        tweet = re.split('\s+', sentence)
        processed_seq = tk.texts_to_sequences(tweet)
        tokens = [inv_map[tok] for seq in processed_seq for tok in seq]
        tokenizeList.append(tokens)

    df['tokens'] = tokenizeList

    return df
```

```
def remove_stopwords(input_text):
    """
    Removes stopwords from tweets based on Indonesian stop word list

    Args:     input_text, string:  tweet

    Returns:  string: cleaned tweet
    """

    stopwords_list = stopwords.words('indonesian')

    # Custom stopwords
    custom_stopwords = ['\n','\n\n', '&amp;', ' ', '.', '-', '$', '@',"rt"]

    words = input_text.split()
    clean_words = [word for word in words if word not in stopwords_list]

    custom_clean_words = [word for word in clean_words if all(cs not in word for
cs in custom_stopwords)]

    return " ".join(custom_clean_words)
```

```
def embedding_to_matrix(token_dict, embeddings, dimensionality, NB_WORDS):
    """
    Converts Pandas df of pre-trained embeddings to a NxN matrix cross-
    referenced with all tokens in corpus.

    Args:     token_dict, dict: dictionary of tokens
              Embeddings, Pandas DataFrame: pre-trained Embeddings
              Dimensionality, int: dimension of embeddings
                                   (i.e. 400-D Word2Vec/100-D FT)
              NB_WORDS, int: Parameter indicating the number of words to
                             put in the embedding dictionary

    Returns:  emb_matrix, NumPy Array:  NxN matrix cross-referenced with
                                        all tokens in corpus
    """

    emb_dict = {}

    for line in embeddings:
        word = line[0]
        vector = np.asarray(line[1:], dtype='float32')
        emb_dict[word] = vector

    emb_matrix = np.zeros((NB_WORDS, dimensionality))

    for w, i in token_dict:
        # Limit to the NB_WORDS
        if i < NB_WORDS:
            vect = emb_dict.get(w)
            # Check if the word from the training data occurs in the
            # pre-trained word embeddings
            # Otherwise the vector is kept with only zeros
            if vect is not None:
                emb_matrix[i] = vect
        else:
            break

    return emb_matrix
```

```
def dict_of_tokens(df, tk):
    """
    Creates a dictionary of tokens found in the tweets.

    Args:     df, Pandas Dataframe: Contains all tweets and labels
              tk, Keras Tokenizer Object:  Creates dict of tokens
                                           via fit_on_texts fn

    Returns:  dictionary of tokens
    """

    tk.fit_on_texts(df['tweet'])

    return tk.word_index.items()
```

```
def convert_text_to_sequences(df, tk):
    """
    Converts tokenized tweets to sequences.

    Args:       df, Pandas Dataframe: Contains all tweets,
                                      tokenized tweets and labels
                tk, Keras Tokenizer Object:  Converts tokenized tweets to
                                             sequences via texts_to_sequences

    Returns:    NumPy array:  List of all tokenized tweets represented as a
                              sequence of numbers
    """

    tokens = df['tokens']

    return tk.texts_to_sequences(tokens)
```

```
def import_embedding(filepath, Dimensionality):
    """
    Imports a pre-trained embedding file and converts to NumPy array.

    Args:       filepath, string:  file path for embedding data
                Dimensionality, int: dimension of embeddings
                                     (i.e. 400-D Word2Vec/100-D FT)

    Returns:    NumPy array:  pre-trained embeddings
    """

    header_list = list(range(0, Dimensionality+1))

    embeddings = pd.read_csv(filepath, delimiter=' ', skiprows=1,
                             index_col=False, names=header_list)

    return embeddings.to_numpy()
```

```
def sequence_pad(X_seq):
    """
    Pads the sequences to some length of max tweet in tokens.

    Args:       X_seq, NumPy array:  List of all tokenized tweets represented
                                     as a sequence of numbers

    Returns:    X_seq_trunc, NumPy array:  List of all tokenized tweets
                                           appended to value of max tweet
                                           length
                MAX_LEN, int:   Maximum tweet length in tokens
    """

    lengths = []

    for i in X_seq:
        lengths.append(len(i))

    MAX_LEN = max(lengths)

    X_seq_trunc = pad_sequences(X_seq, maxlen=MAX_LEN)

    return X_seq_trunc, MAX_LEN
```

```
def encode_labels(df):
    """
    One-hot encodes the emotion labels.

    Args:       df, Pandas DataFrame:  Contains all tweets and labels

    Returns:    y_oh, NumPy array:  one-hot encoded emotion labels
    """

    le = LabelEncoder()

    y = le.fit_transform(df['label'])

    return to_categorical(y)
```

```
def deep_model(model, X_train, y_train, X_valid, y_valid, checkpoint_filepath):
    """
    Function for individual model exploration.
    Saves file at epoch with max validation accuracy.

    Args:       model, Keras object: model with the chosen architecture
                X_train, NumPy array: training features
                y_train, NumPy array: training target
                X_valid, NumPy array: validation features
                Y_valid, NumPy array: validation target
                checkpoint_filepath:  file path to save epoch with max
                                      validation accuracy

    Returns:    model training history, Keras object: contains the output
                of the Keras model.fit fn
    """

    model_checkpoint_callback = keras.callbacks.ModelCheckpoint(
                            filepath=checkpoint_filepath,
                            save_weights_only=False,
                            monitor='val_accuracy',
                            mode='max',
                            save_best_only=True)

    model.compile(optimizer='rmsprop'
                  , loss='categorical_crossentropy'
                  , metrics=['accuracy'])

    history = model.fit(X_train
                        , y_train
                        , epochs=NB_START_EPOCHS
                        , batch_size=BATCH_SIZE
                        , validation_data=(X_valid, y_valid)
                        , verbose=2
                        , callbacks=[model_checkpoint_callback]
                        , shuffle=True )
    return history
```

```
def eval_metric(history, metric_name):
    '''
    Function to evaluate a trained model on a chosen metric.
    Training and validation metric are plotted in a
    line chart for each epoch.

    Args:     history, Keras object: model training history
              metric_name, string: loss or accuracy

    Returns:  matplotlib chart:  line chart with epochs of x-axis and
                                 metric on y-axis
    '''
    metric = history.history[metric_name]
    val_metric = history.history['val_' + metric_name]

    e = range(1, NB_START_EPOCHS + 1)

    plt.plot(e, metric, label='Train ' + metric_name)
    plt.plot(e, val_metric, label='Validation ' + metric_name)
    plt.title('Bi-LSTM ' + metric_name)
    plt.xlabel('Epochs')
    plt.ylabel(metric_name)
    plt.xticks(np.arange(min(e)+1, max(e)+1, 2.0))
    plt.legend()
    plt.show()
```

```
def make_classification_report(X_valid, y_valid, checkpoint_filepath):
    """
    Makes a classification report.

    Args:     X_valid, NumPy array: validation features
              Y_valid, NumPy array: validation target
              checkpoint_filepath:  file path to save epoch with max
                                    validation accuracy

    Returns:  classification report
    """

    model.load_weights(checkpoint_filepath)

    label_names = ["anger", "fear", "joy", "love", "sadness"]

    y_prob = model.predict(X_valid)
    prediction_ints = np.zeros_like(y_prob)
    prediction_ints[np.arange(len(y_prob)), y_prob.argmax(1)] = 1
    prediction = np.where(prediction_ints==1)[1]

    return print(classification_report(y_valid, prediction_ints,
                                       target_names=label_names, digits=4))
```

```
def make_confusion_matrix(X_valid, y_valid, checkpoint_filepath):
    """
    Makes a confusion matrix.

    Args:     X_valid, NumPy array: validation features
              Y_valid, NumPy array: validation target
              checkpoint_filepath:  file path to save epoch with
                                    max validation accuracy

    Returns:  confusion matrix
    """

    model.load_weights(checkpoint_filepath)

    label_names = ["anger", "fear", "joy", "love", "sadness"]

    y_prob = model.predict(X_valid)
    prediction_ints = np.zeros_like(y_prob)
    prediction_ints[np.arange(len(y_prob)), y_prob.argmax(1)] = 1
    prediction = np.where(prediction_ints==1)[1]

    y_cat_valid_emb = np.where(y_valid==1)[1]

    cf_matrix = confusion_matrix(prediction, y_cat_valid_emb)

    cf_matrix_norm = cf_matrix / cf_matrix.astype(np.float).sum(axis=1,
                                                                keepdims=True)

    cf_matrix_norm_round = np.around(cf_matrix_norm, decimals=2)

    df_cm = pd.DataFrame(cf_matrix_norm_round, columns=label_names,
                         index=label_names)

    df_cm.index.name = 'Actual'
    df_cm.columns.name = 'Predicted'

    sns.heatmap(df_cm, cmap='Blues', annot=True)
    plt.yticks(rotation=0)
    plt.title("Confusion Matrix")
    plt.show()
```

```
NB_WORDS = 20000  # max number of words to be put in the dictionary
NB_START_EPOCHS = 20  # Number of epochs used for training
BATCH_SIZE = 64  # Batch size used in the mini-batch gradient descent
MAX_LEN = 64  # Maximum number of words in a sequence
WORD2VEC_DIM = 400  # Dimension of the pre-trained word2vec embeddings
FASTEXT_DIM = 100 # Dimension of the pre-trained fasttext embeddings
NUM_FOLDS = 10 # Number of folds in K-Fold Cross Validation experiment

checkpoint_filepath = 'insert file path here'
tweets_filepath = 'insert file path here'
word2vec_filepath = 'insert file path here'
fastext_filepath = 'insert file path here'

TRAIN_SIZE = .9
VALID_SIZE = .1
```

```
# import tweets
df = pd.read_csv(tweets_filepath)

# import word2vec pre-trained embeddings
w2v_embeddings = import_embedding(word2vec_filepath, WORD2VEC_DIM)

# import fasttext pre-trained embeddings
ft_embeddings = import_embedding(fastext_filepath, FASTEXT_DIM)
```

```
# initialize the tokenizer
tk = Tokenizer(num_words=NB_WORDS,
               filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~\t\n',
               lower=True,
               split=" ")

# tokenize tweets and remove stopwords
df = custom_tokenizer(df, tk)
```

```
# label encoding
y_oh = encode_labels(df)
```

```
# create token dictionary
token_dict = dict_of_tokens(df, tk)
```

```
# convert data to numerical sequences
X_seq = convert_text_to_sequences(df, tk)
```

```
# create word sequences of equal length (MAX_LEN)
X_seq_trunc, MAX_LEN = sequence_pad(X_seq)
```

```
# create embedding matrices
w2v_emb_matrix = embedding_to_matrix(token_dict, w2v_embeddings,
                                     WORD2VEC_DIM, NB_WORDS)

ft_emb_matrix = embedding_to_matrix(token_dict, ft_embeddings,
                                    FASTEXT_DIM, NB_WORDS)
```

```
# cross validation experiment
# RNN Variants can be toggled by uncommenting/commenting
# the model.add(LSTM/GRU/etc.) line

kf = KFold(n_splits=NUM_FOLDS)

i=0

for train_index, test_index in kf.split(X_seq_trunc):
    i+=1
    print("FOLD", i)
    X_train, X_test = X_seq_trunc[train_index], X_seq_trunc[test_index]
    y_train, y_test = y_oh[train_index], y_oh[test_index]

    model = Sequential()
    model.add(Embedding(NB_WORDS, FASTEXT_DIM, input_length=MAX_LEN))
    model.add(SpatialDropout1D(0.2))
    model.add(Bidirectional(LSTM(512, dropout=0.2, recurrent_dropout=0.2,
                                 return_sequences=True)))
    #model.add(Bidirectional(LSTM(256, dropout=0.2, recurrent_dropout=0.2,
    #                             return_sequences=True)))
    #model.add(LSTM(512, dropout=0.2, recurrent_dropout=0.2,
    #                             return_sequences=True))
    #model.add(GRU(512, dropout=0.2, recurrent_dropout=0.2,
    #                             return_sequences=True))
    model.add(GlobalMaxPool1D())
    model.add(Dense(512, activation="relu"))
    model.add(layers.Dropout(rate=0.5))
    model.add(Dense(256, activation="relu"))
    model.add(layers.Dropout(rate=0.5))
    model.add(layers.Dense(5, activation='softmax'))
    model.layers[0].set_weights([ft_emb_matrix])
    model.layers[0].trainable = True
    print(model.summary())

    model.compile(loss="categorical_crossentropy",
                  optimizer="rmsprop", metrics=["acc"])

    model.fit(X_train, y_train,
              validation_data=(X_test, y_test),
              epochs=NB_START_EPOCHS, batch_size=BATCH_SIZE, verbose=2)
```

```
# bootstrap experiment
# RNN Variants can be toggled by uncommenting/commenting
# the model.add(LSTM/GRU/etc.) line

bootdf = pd.DataFrame(X_seq_trunc)

bootdf['label'] = df['label']

for i in range(NUM_FOLDS):

    print("FOLD", i)

    # create a training set that is 90% of the data
    train = resample(bootdf, replace=True, n_samples=3960, random_state=i)
    test = bootdf[~bootdf.index.isin(train.index)]

    X_train = train.iloc[:,:-1]
    X_test = test.iloc[:,:-1]

    y_train = train.iloc[:, -1]
    y_test = test.iloc[:, -1]

    le = LabelEncoder()
    y = le.fit_transform(y_train)
    y_train = to_categorical(y)

    y2 = le.fit_transform(y_test)
    y_test = to_categorical(y2)

    model = Sequential()
    model.add(Embedding(NB_WORDS, FASTEXT_DIM, input_length=MAX_LEN))
    model.add(SpatialDropout1D(0.2))
    model.add(Bidirectional(LSTM(256, dropout=0.2, recurrent_dropout=0.2,
                                 return_sequences=True)))
    #model.add(Bidirectional(LSTM(256, dropout=0.2, recurrent_dropout=0.2,
    #                             return_sequences=True)))
    #model.add(LSTM(512, dropout=0.2, recurrent_dropout=0.2,
    #                             return_sequences=True))
    #model.add(GRU(512, dropout=0.2, recurrent_dropout=0.2,
    #                             return_sequences=True))
    model.add(GlobalMaxPool1D())
    model.add(Dense(512, activation="relu"))
    model.add(layers.Dropout(rate=0.5))
    model.add(Dense(256, activation="relu"))
    model.add(layers.Dropout(rate=0.5))
    model.add(layers.Dense(5, activation='softmax'))
    model.layers[0].set_weights([ft_emb_matrix])
    model.layers[0].trainable = True
    print(model.summary())

    model.compile(loss="categorical_crossentropy",
                optimizer="rmsprop", metrics=["acc"])

    model.fit(X_train, y_train,
            validation_data=(X_test, y_test),
            epochs=NB_START_EPOCHS, batch_size=BATCH_SIZE, verbose=2)
```

```
# train/test split
X_train, X_test, y_train, y_test = train_test_split(X_seq_trunc, y_oh,
                                                    test_size=VALID_SIZE,
                                                    random_state=0)
```

```
# individual model exploration
model = Sequential()
model.add(Embedding(NB_WORDS, FASTEXT_DIM, input_length=MAX_LEN))
model.add(SpatialDropout1D(0.2))
model.add(Bidirectional(LSTM(512, dropout=0.2, recurrent_dropout=0.2,
                             return_sequences=True)))
model.add(GlobalMaxPool1D())
model.add(Dense(512, activation="relu"))
model.add(layers.Dropout(rate=0.5))
model.add(Dense(256, activation="relu"))
model.add(layers.Dropout(rate=0.5))
model.add(layers.Dense(5, activation='softmax'))
model.layers[0].set_weights([ft_emb_matrix])
model.layers[0].trainable = True
model_history = deep_model(model, X_train, y_train, X_test, y_test, checkpoint_f
ilepath)
print(model.summary())
```

```
# model accuracy over epochs
eval_metric(model_history, 'accuracy')
```

```
# model loss over epochs
eval_metric(model_history, 'loss')
```

```
# model classification report
make_classification_report(X_test, y_test, checkpoint_filepath)
```

```
# model confusion matrix
make_confusion_matrix(X_test, y_test, checkpoint_filepath)
```

Appendix B. Aggregate Cross Validation Analysis

Each of the four RNN variant models' aggregate cross validation confusion matrices can be seen below in Figures 21 through 24. These models were examined to determine if a model other than the single-layer Bi-LSTM displayed better performance in the fear and anger classes. This analysis was performed in case a model with lesser overall F1 score would be preferable to an operational user due to exceptional performance in the classes deemed most important for a military intelligence context. However, these confusion matrices show that each model displays similar behavior when classifying these classes. Therefore, this research retains it's finding that the single-layer Bi-LSTM is optimal.

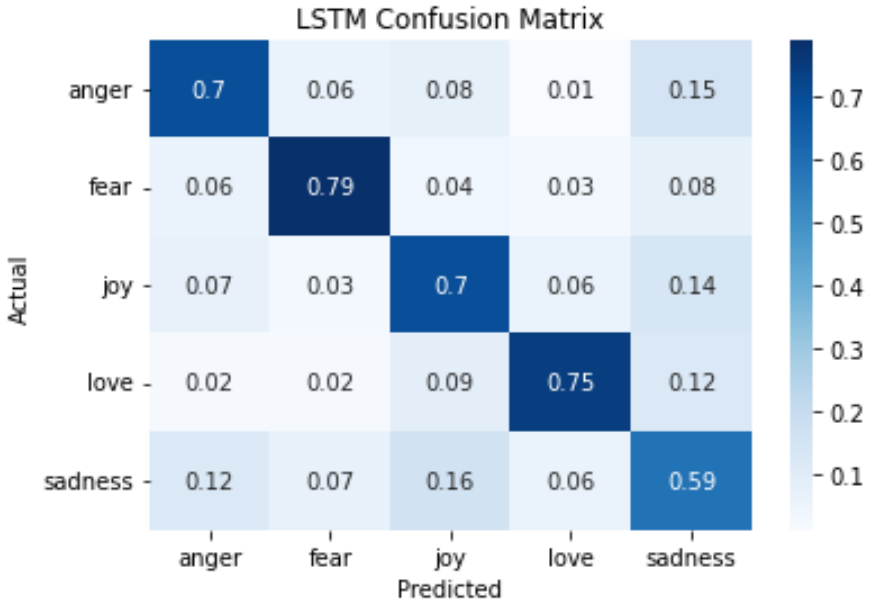

Figure 21: Aggregate Cross Validation LSTM Confusion Matrix

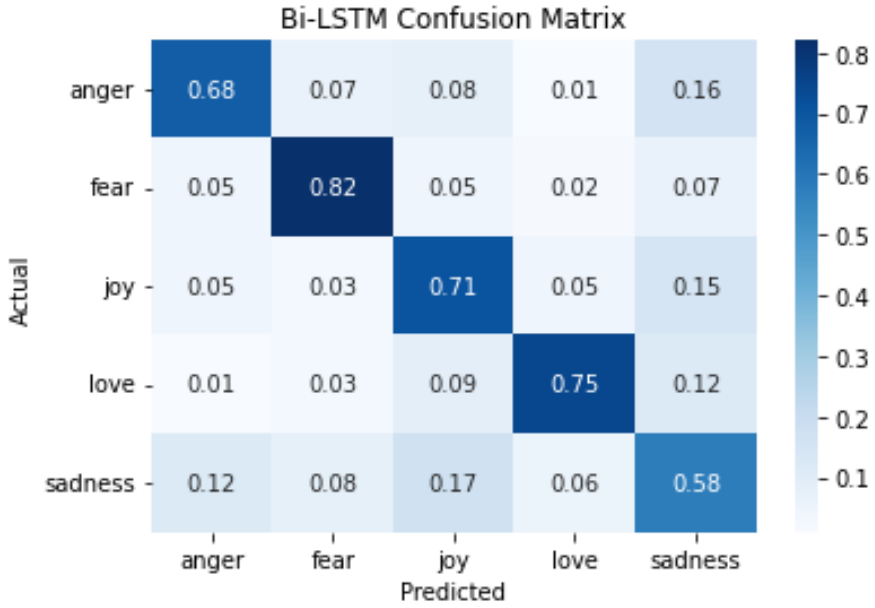

Figure 22: Aggregate Cross Validation Bi-LSTM Confusion Matrix

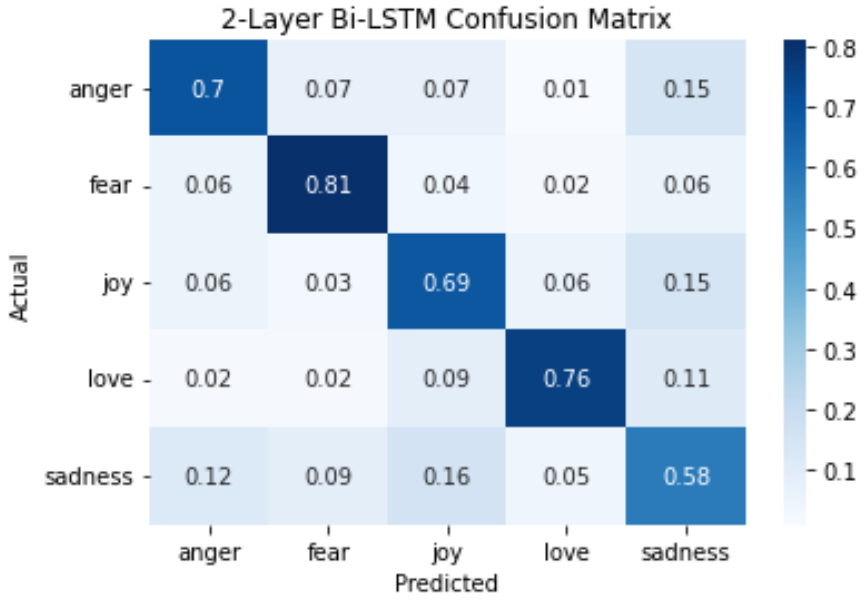

Figure 23: Aggregate Cross Validation Stacked Bi-LSTM Confusion Matrix

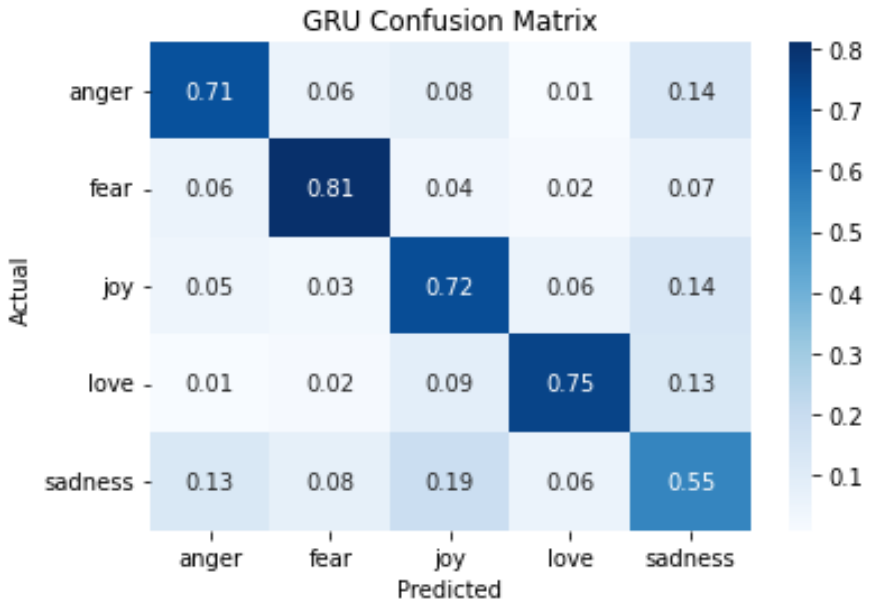

Figure 24: Aggregate Cross Validation GRU Confusion Matrix

Bibliography

Bartles, Charles. "Getting Gerasimov Right." *Military Review*, (January-February): 30–38, 2016.

Beskow, David M. and Kathleen M. Carley. "Social Cybersecurity An Emerging National Security Requirement." *Military Review*, 99:117, 2019.

Cambria, Erik and Bebo White. "Jumping NLP Curves: A Review of Natural Language Processing Research." *IEEE Computational Intelligence Magazine*, 9(2): 48–57, 2014.

Carley, Kathleen M., Guido Cervone, Nitin Agarwal, and Huan Liu. "Social Cyber-Security." *Social Cultural, and Behavioral Modeling: 11th International Conference, SBP-BRiMS 2018, Washington, DC, USA, July 10-13, 2018*, 389–394, New York. Springer, 2018.

Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling." *NIPS 2014 Workshop on Deep Learning*, 2014.

Ciftci, Basri and Mehmet Serkan Apaydin. "A Deep Learning Approach to Sentiment Analysis in Turkish." *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*, 1–5. IEEE, 2018.

Department of Defense. "Summary of the 2018 National Defense Strategy of the United States of America," 2018.

Devi, S., K. Naveenkumar, S. Shakthi Ganesh, and S. Ritesh. "Location Based Twitter Emotion Classification for Disaster Management." *2021 Third International*

Conference on Inventive Research in Computing Applications (ICIRCA), 664–669. IEEE, 2021.

Elfaik, Hanane and El Habib Nfaoui. "Deep Bidirectional LSTM Network Learning-Based Sentiment Analysis for Arabic Text." *Journal of Intelligent Systems*, 30(1): 395–412, 2020.

Garg, Nikhil, Londa Schiebinger, Dan Jurafsky, and James Zou. "Word Embeddings Quantify 100 Years of Gender and Ethnic Stereotypes." *Proceedings of the National Academy of Science*, 115(16):E3635–E3644, 2018.

Gavrilis, James A. "A Model for Population-Centered Warfare: A Conceptual Framework for Analyzing and Understanding the Theory and Practice of Insurgency and Counterinsurgency." *Small Wars Journal*, 1–20, 2009.

Géron, Aurélien. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media Inc., 2nd edition, 2019.

Haines, Julia M. "Automated Sentiment Analysis for Personnel Survey Data in the US Air Force Context." *Theses and Dissertations*, (4927), 2021.

Hassan, Abdalraouf and Ausif Mahmood. "Deep learning for sentence classification." *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, 1–5. IEEE, 2017.

Hochreiter, Sepp and Jürgen Schmidhuber. "Long Short-Term Memory." *Neural Computation*, 9(8):1735–1780, 1997.

Imaduddin, Helmi, Widyawan, and Silmi Fauziati. "Word Embedding Comparison for Indonesian Language Sentiment Analysis." *2019 International Conference of Artificial Intelligence and Information Technology (ICAIIT)*, 426–430. IEEE, 2019.

Indulkar, Yash and Abhijit Patil. "Comparative Study of Machine Learning Algorithms for Twitter Sentiment Analysis." *2021 International Conference on Emerging Smart Computing and Informatics*, 295–299, New York. IEEE Press, 2021.

Jagtap, V. S and Karishma Pawar. "Analysis of different approaches to Sentence-Level Sentiment Classification." *International Journal of Scientific Engineering and Technology*, 2(3), 2013.

Jipa, Gabriel. "The Value of Structured and Unstructured Content Analytics of Employees' Opinion Mining." *Journal of Administrative Sciences and Technology*, 2019:1–25, 2019.

Jurafksy, Daniel and James H. Martin. *Speech and Language Processing*. 3rd (draft) edition, 2020.

Koto, Fajri and Gemala Y. Rahmaningtyas. "Inset Lexicon: Evaluation of a Word List for Indonesian Sentiment Analysis in Microblogs." *2017 International Conference on Asian Language Processing*, 391–394, New York. IEEE Press, 2017.

Lailiyah, M., S. Sumpeno, and I. K. E. Purnama. "Sentiment Analysis of Public Complaints Using Lexical Resources Between Indonesian Sentiment Lexicon and Sentiwordnet." *2017 International Seminar on Intelligent Technology and Its Applications*, 307–312, New York. IEEE Press, 2017.

Liang, Xiao, Zhiming Liu, and Chunping Ouyang. "A Multi-Sentiment Classifier Based on GRU and Attention Mechanism." *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, 527–530. IEEE, 2018.

Manning, Charles. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

McHugh, Mary L. "Interrater Reliability: the kappa statistic." *Biochemia medica*, 22 (3):276–282, 2012.

Nimmo, Ben. "Anatomy of an Info-War: How Russia's Propaganda Machine Works, and How to Counter It." *Central European Policy Institute*, 2015.

Raisa, Jasiya Fairiz, Maliha Ulfat, Abdullah Al Mueed, and S. M. Salim Reza. "A Review on Twitter Sentiment Analysis Approaches." *2021 International Conference on Information and Communication Technology for Sustainable Development*, 375–379, New York. IEEE Press, 2021.

Raschka, Sebastian. "Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning," 2018. URL `https://arxiv.org/abs/1811.12808`.

Ribeiro, Filipe Nunes, Matheus Araújo, Pollyanna Gonçalves, Fabrício Benevenuto, and Marcos André Gonçalves. "SentiBench - a benchmark comparison of state-of-the-practice sentiment analysis methods." *EJP Data Science*, 5(1), 2015.

Rugge, Fabio. "'Mind Hacking': Information Warfare in the Cyber Age." *Italian Institute for International Political Studies*, 319:1–8, 2018.

Saputri, Mei Silviana, Rahmad Mahendra, and Mirna Adriani. "Emotion Classification on Indonesian Twitter Dataset." *Proceeding of International Conference on Asian Language Processing*, 90–95, 2018.

Semeniuta, Stanislau, Aliaksei Severyn, and Erhardt Barth. "Recurrent Dropout without Memory Loss," 2016. URL `https://arxiv.org/abs/1603.05118`.

Smithmeyer, Colby. "G2 Analytics Kickoff." Technical report, The Research and Analysis Center-Monterey, 2021.

Xiao, Zheng and PiJun Liang. "Chinese Sentiment Analysis Using Bidirectional LSTM with Word Embedding." *International Conference on Cloud Computing and Security*, 601–610, 2016.

Zhou, Junhao, Yue Lu, Hong-Ning Dai, Hao Wang, and Hong Xiao. "Sentiment Analysis of Chinese Microblog Based on Stacked Bidirectional LSTM." *2018 15th International Symposium on Pervasive Systems, Algorithms and Networks (I-SPAN)*, 162–167. IEEE, 2018.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704–0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
24–03–2022	Master's Thesis	Sept 2020 — Mar 2022

4. TITLE AND SUBTITLE

Narrative Analysis of Open-Source
Social Media Activity in the INDOPACOM AOR

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Glenn, Aaron K, Capt, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENS-MS-22-M-131

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

The Research and Analysis Center-Monterey
LTC Brian M. Wade, Director
1 University Circle
Monterey CA 93943
(831) 656-3086
Email: brian.wade@nps.edu

10. SPONSOR/MONITOR'S ACRONYM(S)

TRAC-Monterey

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

This work is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

Emotion classification can be a powerful tool to derive narratives from social media data. Recurrent Neural Networks (RNN) can meet or exceed the performance of state-of-the-art traditional machine learning techniques using exclusively open-source data and models. Specifically, these results show that RNN variants can produce more than an 8% gain in accuracy in comparison to Logistic Regression and SVM techniques and a 15% gain over Random Forest when using FastText embeddings. This research found a statistical significance in the performance of a single layer Bi-directional Long Short-Term Memory (Bi-LSTM) model over a 2-layer stacked Bi-LSTM model. This research also found that a single layer Bi-LSTM RNN met the performance of a state-of-the-art Logistic Regression model with supplemental closed-source features from a study by Saputri et al. (2018) when classifying the emotion of Indonesian Tweets. This model can be provided to operational units within the INDOPACOM theater giving them the ability to identify social media posts based on predicted emotion class - allowing them to gauge public reaction to military exercises in theater.

15. SUBJECT TERMS

emotion classification, long short-term memory (LSTM), machine learning, natural language processing (NLP), recurrent neural network (RNN)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			LTC Phillip M. LaCasse Ph.D, AFIT/ENS
U	U	U	UU	81	19b. TELEPHONE NUMBER *(include area code)* (937) 255-6565 x4318; phillip.lacasse@afit.edu