An Approach to Email Spam Classification Through a Long Short-Term Memory Neural Network

Andrew Yin Li
Oberlin College
Oberlin, OH
ali@oberlin.edu

Joshua Rappaport
Oberlin College
Oberlin, OH
jsrappaport@gmail.com

Benjamin McGarvey
Oberlin College
Oberlin, OH
bmcgarve@oberlin.edu

Abstract—**Naive Bayes has long been favored as the traditional approach to filtering spam due to the algorithms ease of implementation and a relatively high accuracy coupled with a low overhead. However, the recent emergence of graphical processing unit architecture capable of supporting neural networks with sizable inputs has led to the prevalence of various neural networks in spam detection and filtering. In this paper, we delve into a novel approach to detecting spam through a Long Short-Term Memory (LSTM) neural network implemented through Keras and compare the results of our implementation to results yielded by our own implementation of Multinomial Naive Bayes through scikit-learn on the SpamAssassin email corpus in order to determine the feasibility of LSTM neural networks in spam detection. We discover that while our LSTM neural network was substantially slower than the Naive Bayes model, our LSTM neural network was able to achieve a slightly higher accuracy.**

I. Introduction

An ever-present privacy and security risk, spam emails accounted for 56.63% of emails in 2017, with 13.21% of spam emails originating from the United States alone. Moreover, spam has adapted according to contemporary trends to better target victims [4]. For example, there has been a rise in cryptocurrency-related spam emails, predominantly containing cryptocurrency scams, distribution of malware that mines cryptocurrency in the background of your computer, as well as the distribution of ransomware demanding payments in cryptocurrency. The prevalence of spam and malicious emails in general is concerning, hence the necessity to constantly develop better techniques to filter spam [4], [6].

While Bayesian algorithms were already prominent in spam filtering [11], Paul Grahams 2002 essay set the foundation of modern day spam filter implementations, popularizing the utilization of the Naive Bayes classifier and Bag of Words feature selection due to the methods ease of implementation and a relatively high classification accuracy coupled with a low overhead [3]. Furthermore, implementations of Naive Bayes and feature selection algorithms have become standardized in machine learning libraries, such as the scikit-learn library used in this paper, making Naive Bayes models even easier to implement [10]. However, the comparatively low runtime and ease of implementation comes at the expense of a slightly lower accuracy when compared to high runtime and complex classifiers such as neural networks. Recent substantial strides in graphical processing unit architecture has enabled feasible implementations of neural networks with considerably large inputs, which in turn has led to the emergence of experimentation with neural networks in spam detection and filtering given the large quantity of emails required to train neural network models [5]. Convolutional and recurrent neural networks are typically used for spam detection and have been proven to be slightly more accurate than Naive Bayes at filtering spam, but with a significantly higher runtime [7]–[9]. The study detailed by this paper was conceived in an attempt to examine a novel approach to filtering spam. We chose to implement a Long Short-Term Memory (LSTM) neural network spam filter because we could not find any scholarly papers nor implementations on such a topic. To substantiate our study, we chose Multinomial Naive Bayes to be our baseline classifier to use as a comparison to our LSTM neural network. We specifically sought to address the following questions:

- Are LSTM neural networks a feasible approach to combating spam?
- Can we prove that a LSTM neural network is more accurate than a Naive Bayes classifier at detecting spam emails?
- If so, to what extent does this hold and what are the constraints of the neural network in comparison to Naive Bayes?

We theorized through intuition gathered from our background research [6]–[9] that a LSTM neural network would be slightly more accurate than Multinomial Naive Bayes at detecting spam emails, albeit substantially slower.

Using a preprocessor of our own design, we parsed spam and ham emails from the publicly available SpamAssassin corpus and fed them either into our tokenizer and then into our LSTM neural network and our Multinomial Naive Bayes model utilizing vectorized word count feature extraction. We discover that our hypothesis was indeed correct and discuss our findings and their implications, as well as future avenues of research.

II. Background Research from Related Work

Hochreiter and Schmidhuber wrote the original paper introducing the LSTM concept as a solution to the problem of time lag in recurrent neural networks. This work allowed neural networks to analyze much longer data streams, a critical issue that would have otherwise rendered Recursive

Neural Networks as an impossible solution to the email spam problem.

Olah wrote a fantastically detailed blog post about LSTM networks and explored their usefulness with regards to analyzing text.

A. Recurrent Neural Networks

Our initial idea was to use implement a generic Recurrent Neural Network (RNN) to filter spam. RNNs attempt to solve problems by reading over a set of features one by one, outputting at each step the networks guess at a solution. When a feature set is read over completely, the network outputs the final guess. The ingenuity of the RNN is that at each step, the output is fed back into itself as a parameter to analyze the next feature. [1] Over time, the network is able to learn relationships between features in sequence. This is very beneficial when working with data sets where the order of features is relevant to the solution. [1] **For example, this bold sentence you are reading right now can be understood as a series of features (words) in an ordered sequence.** Proper understanding of the sentence is not limited to noting the number of times ordered, understood, features and sentence appear. Correct analysis takes into account the ordering of the words as well. Although a RNN will only output the result after exhausting all features in a single selection, every time step it is calculating a guess for the selection. These guesses are fed back into the network during the next time step, and are used in calculating the next guess. However, after reading a plethora of papers on RNN implementations of spam filters, we realized we would merely be reinventing the wheel if we chose to proceed with an RNN implementation in our study [7].

B. Long Short-Term Memory Neural Networks

Long Short-Term Memory (LSTM) networks are an evolution of the traditional RNN. The basic principle of both is the same: The network learns relationships between features, and is able to base its decisions on the order of features, not just on their presence or absence.

LSTM networks solve an unexpected problem with RNNs. Traditional RNNs failed when the space between key features was too great. [2] For example, when analyzing the text of a child talking about themselves, they might start off the paragraph saying I was born in England. They might go on and on for a while, listing facts about their dad, mom, their pets, what they like to do after school, before finally saying, I speak-. We expect that the next word in the sentence will be English because they live in England, but since the gap between the present and the time they said that they lived in England is so wide, it is unlikely for the RNN agent to correctly identify the next word. [1]

LSTM networks solve that problem by introducing the idea of state. A single agent keeps track of their own state (a vector array) and updates it during every time increment on a given piece. [2] Then, when calculating the next output, it takes into account the given input, the last output, and the state it was in after the last time step. So, on every time step, the network receives a feature x, the last output y_{t-1}, and the state S_{t-1}. It then outputs a new output y_t and an updated state S_t. Internal to the network are W (a vector space) and b (a value). Both are set by the LSTM itself and are updated through backpropagation.

The following is a brief breakdown of the calculations that occur at each time step. [1][1]

1) Given x and y_{t-1}, the network calculates how much of each vector in the state S_{t-1} to forget.
$f_t = \sigma(W_f \cdot [y_{t-1}, x_t] + b_f)$

2) Next, it decides what to remember. A sigmoid function i_t (the input gate layer) decides which vectors in the state to update, and a tanh function $\tilde{S}_t$ decides what values to remember, and the scale of importance of those new values. Then the old state values are forgotten and the new ones are added to create S_t.
$i_t = \sigma(W_i \cdot [y_{t-1}, x_t] + b_i)$
$\tilde{S}_t = tanh(W_s \cdot [y_{t-1}, x_t] + b_s)$
$S_t = f_t \cdot S_{t-1} + i_t \cdot \tilde{S}_t$

3) Finally, the network decides what to output. It decides what sections of the cell state to output o_t and then runs the cell state S_t through a tanh function (so our values are all between -1 and 1). Then these two values are multiplied together and submitted as output.
$o_t = (W_o \cdot [y_{t-1}, x_t] + b_o)$
$y_t = o_t \cdot tanh(S_t)$

C. Multinomial Naive Bayes

Given the rationale that any neural network will have a higher runtime than Naive Bayes when trained on the same dataset, it follows that if our LSTM neural network had a lower classification accuracy than Naive Bayes, it would not be worth comparing to other neural networks, hence our decision to use Naive Bayes as a baseline. Naive Bayes derives rules through the “application of the Bayes theorem through the ‘naive assumption’ of independence between every combination of features” [10]. Given a label y, features x_1 through x_n, and $P(a \mid b)$ indicating the probability of a given b, we ultimately arrive at:

$$\text{Correct label } y_* = argmax_y P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

Variations of Naive Bayes calculate $P(a \mid b)$ differently. We ultimately chose to use the scikit-learn library’s Multinomial Naive Bayes implementation due to the prevalence of Multinomial Naive Bayes in text classification and spam email filtering in academic papers and studies we read.

$P(a \mid b)$ is calculated by Multinomial Naive Bayes as follows according to scikit-learn’s documentation:

$$P(a \mid b) = \frac{N_{yi} + \alpha}{N_y + n\alpha}$$

[1] Much of this information is expanded upon in the cited blog post.

N_{yi} is is the number of occurrences of feature i in the training set where the label is y, N_y is the number of occurrences of all features in the training set, n is the number of features, and α is a "smoothing" constant left at 1 for our purposes [10].

III. Methodology

Our study was conducted in the Python language, predominantly due to the scope of our implementations requiring machine learning libraries, making Python the obvious choice due to utility offered by the scikit-learn and TensorFlow libraries.

A. Data Set and Preprocessing

We decided to use the SpamAssassin email corpus after seeing its use in a related study [12]. We used a set of about 3000 emails with all features intact, divided into categories of spam and ham (not-spam, i.e. normal emails). There is a distribution of approximately 500 spam and 2500 ham in this corpus, determined by what was available from the corpus. We believe that the majority of emails you receive are indeed ham, hence the 1:4 ratio of spam to ham. We chose to use the ham corpus titled "easy ham" to speed up training of the neural network. Our preprocessor takes two arguments– a directory filled with spam emails in .txt format and a directory filled with ham emails following suit. The preprocessor processes emails from each directory individually, stemming words to their roots (e.g. kicks, kicked, kicker all stem to kick) and stripping words of non-alphanumeric characters through a regular expression, before creating the stemmed version of the email in a new directory containing all stemmed spam and ham emails. This directory is then passed to either our Naive Bayes model or our tokenizer followed by the neural network.

B. Multinomial Naive Bayes Implementation

Our implementation requires user input of the path to the directory containing stemmed spam and ham emails, a seed used to determine the random splitting of the stemmed emails into training and testing sets, and the percentage of the stemmed emails to use for training. The program splits the stemmed emails into training and testing sets according to the arguments passed to the program, before generating a vocabulary containing the 2000 most common words across all training emails. The model is trained by a matrix containing word count vectors for each email, in which each row corresponds to an email and each column maps to the corresponding column in the vocabulary and contains the word count in the email for the word it maps to.

C. Tokenizer and Long Short-Term Memory Neural Network Implementation

Before we began the implementation of the neural network we needed to add one extra step to the preprocessing of the emails. Because the LSTM network takes as input an array of integers, all of the emails needed to be tokenized. We created array representations of the spam and ham email sets, as well as a tokenizer object to tokenize each word on the fly. The tokenizer takes in individual words and returns the index of that word.

The LSTM network is a one-layer network built using Keras, a TensorFlow API that allows for quick and easy modification of a network's model. The network we built is a simple three layer model. The first layer is an embedding layer that embeds each word into a matrix of size 32*L where L is the length of the vocabulary. The second layer is a set of 300 LSTM agents. Finally, there is a Dense layer that calculates the most likely classification of the email and outputs either a 1 or 0 depending on if it is spam or ham respectively. We used Adam as our backpropogation algorithm.We trained the model on 75% of the data set and tested it on the remaining 25%. The network was trained with a batch size of 32, and tested after 3 epochs using a variety of different seeds.

IV. Results

A. Naive Bayes

Confusion Matrix 1: Average Naive Bayes Results

		Predicted	
		Spam	**Ham**
Actual	**Spam**	TP 111.8	FN 11.3
	Ham	FP 1.5	TN 638.4

TABLE I
Multinomial Naive Bayes Model Statistics

Metric	***Percentage***
Trials, train : test ratio	30, 3:1
Average runtime	3m34s
Highest accuracy	99.1%
Lowest accuracy	97.6%
Average accuracy	98.3%
Average precision	99.5%
Average sensitivity	90.8%
Average specificity	99.8%

B. Long Short-Term Memory Neural Network

Confusion Matrix 2: LSTM Neural Network Results

		Predicted	
		Spam	**Ham**
Actual	**Spam**	TP 120.4	FN 5.6
	Ham	FP 1.2	TN 636.8

TABLE II
LSTM NEURAL NETWORK MODEL STATISTICS

Metric	***Percentage***
Trials, train : test ratio	20, 3:1
Average runtime	26m42s
Highest accuracy	99.48%
Lowest accuracy	98.82%
Average accuracy	99.1%
Average precision	99.0%
Average sensitivity	95.5%
Average specificity	99.8%

V. ANALYSIS

A. Implications

Through Confusion Matrix 1 and Table 1, we find that our Naive Bayes accuracy is comparable to the Naive Bayes numbers exhibited in our background research and thus satisfactory, as we can assume that we have a reliable baseline for our study. Moreover, the average accuracies and runtimes of Naive Bayes in Table 1 and the LSTM Neural Network in Table 2 seem to confirm our hypothesis; while our Naive Bayes implementation was substantially faster than our LSTM neural network implementation, the neural network had a slightly higher accuracy, with an average accuracy of 99.1% over Naive Bayes' average of 98.3%. Furthermore, the neural network's lowest accuracy of 98.82% was greater than Naive Bayes' accuracy, further indicating that LSTM neural networks are likely more accurate in general than Naive Bayes spam filters at detecting spam.

Comparing Table 1 and Table 2, we find that the LSTM neural network beat Naive Bayes in almost every metric, having a higher runtime as expected, but averaging about the same in specificity. Surprisingly, we expected the LSTM neural network to not have a lower sensitivity percentage in comparison to its other metrics after noticing the disparity between Naive Bayes' other metrics hovering in the 97-99% range and an average sensitivity at 90.8% given the "short-term memory" facet of LSTM neural networks, but were rather astonished to discover the contrary. While the LSTM neural network managed a sensitivity of 95.5%, substantially higher than that of the Naive Bayes implementation, both models exhibited a low average false positive rate at 1.5 and 1.2 respectively, as evident in Confusion Matrix 1 and 2, but a comparatively higher false negative rate, with Naive Bayes averaging 11.3 false negatives and the LSTM neural network averaging 5.6 false negatives. We attribute this phenomenon to the disparate ratio of spam to ham in our data set; at 1:4, it follows that both models "know" that there is more ham than spam and thus predict ham more often.

Beyond our LSTM neural network having a higher accuracy than Naive Bayes, the runtime of the LSTM neural network was the second most prominent metric in this study, with Naive Bayes finishing in under 4 minutes at 3 minutes 34 seconds, whereas the LSTM neural network required 26 minutes and 42 seconds to just complete 3 epochs. While not thoroughly documented in this paper, our experimentation led us to postulate that 6 epochs were necessary to reach 99% accuracy on every seed, but the runtime always exceeded 45 minutes, which led to us solely testing accuracy at 3 epochs because the accuracy average 99.1% and never dipped below 98.82%. We felt that the half hour trade-off in exchange for less than half a percent more accuracy was not worth it, which begs the larger question whether a consumer would be willing to run a spam filter for approximately 23 minutes longer given the constraints of our study, just to achieve 0.8% higher filtering accuracy.

B. Limitations and a Minor Inconsistency

Given the sample size of only approximately 3000 emails from 2004 , with about 500 spam and 2500 easy ham (to expedite the training of the neural network), we tentatively assert that LSTM neural networks are more accurate than generic Multinomial Naive Bayes models, as we are not sure how a small-ish, older data set with obvious instances of ham factor into the overall accuracy of the model; in fact, Naive Bayes could potentially to be more accurate than LSTM neural networks on newer sets with less conspicuous emails, but we doubt that given that we are under the impression that neural networks are frequently used to solve problems in which subtle differences matter.

The extent of our study was performed on a high wattage quad-core mobile CPU instead of a GPU– Keras is built on top of TensorFLow which meant that we could have ran the network on a GPU for theoretically a much lower runtime but time constraints prevented us from experimenting on a GPU. On a high-end GPU, the runtime could be low enough that the aforementioned tradeoff of higher runtime for filtering accuracy could be alleviated to the point that use of a LSTM neural network for spam detection could be as justifiable as using a Naive Bayes-based spam filter. We noted during the analysis of our results that the total entries in Confusion Matrix 1 and 2 did not add up; our Naive Bayes implementation had a test set with 763 elements and our LSTM neural network had a test set with 764 elements. We determined that our Naive Bayes implementation calculated the training set by flooring the product of the training percentage and the overall quantity of emails, whereas the neural network did not, leading to this minor inconsistency. While changing this small detail would slightly alter our results, we believe that it is not significant enough of an issue to invalidate the findings of this work.

C. Future Work

It would be beneficial to fix our slight inconsistency with separating testing and training sets in order to facilitate a better comparison between models. We are also assessing the real time elapsed through Python's time library rather than measuring the time the program actually requires to finish running; we're currently accounting for arbitrary background processes. We could also develop an intuitive GUI wrapper to demonstrate the capabilities of a LSTM neural network in spam filtering and better facilitate distribution on to allow for more widespread experimentation without a significant amount of technical experience or effort compiling the code from source.

On a hardware level, it is imperative that we run our neural network through CUDA on a high-end NVIDIA GPU to see if our network has any current applications in, for example, a corporate environment where one has access to such resources. Moreover, if we were able to run our network on several different GPU models, we could come up with an accurate benchmark to assess hardware requirements for the LSTM neural network to run at an appropriate time in which the trade-off of time for accuracy would be more desirable than not. In a similar vein, it would also be beneficial to run our neural network on a larger, newer corpus or even real emails to see how the neural network fairs in comparison to a Naive Bayes implementation

Finally, our initial rationale behind comparing a LSTM neural network spam filter to a Naive Bayesian one instead of another neural network was that it had already been proven that other neural networks were more accurate but ran slower, which followed that a LSTM neural network would likely run slower and if it were less accurate that Naive Bayes, then it would not be worth comparing to other neural networks. Now that we are fairly certain that a LSTM neural network spam filter is more accurate than using Naive Bayes to filter spam, we would like to take advance our study even further and evaluate our implementation of a LSTM neural network spam filter in comparison to a convolutional neural network spam filter and determine whether there is a conspicuously superior network, or if there is a trade-off to be observed or a network performing better at detecting either spam or ham than other networks.

VI. Conclusion

Our study on applying a LSTM neural network spam filter to the SpamAssassin corpus provides evidence that LSTM neural networks are more accurate at detecting spam than a Naive Bayes spam filter as our LSTM neural network spam filter achieved an average accuracy of 99.1% and a minimum accuracy of 98.82%, surpassing the Naive Bayes spam filter's average of a 98.3% accuracy. As expected, this slight increase in accuracy comes at the cost of greater overhead for training the neural network. We make the assertion that a LSTM neural network spam filter is more accurate than a Naive Bayes spam filter with confidence, but also with caution as further testing with a larger, newer corpus of emails is required to fully confirm and substantiate our findings given that we cannot find other LSTM neural network spam filters to fully confirm and supplement our findings. Additionally, we believe that currently in an environment where there is little margin for error, our neural network is a feasible approach to combating spam, as security has priority over expediency in those scenarios. Moreover, with hardware advancements the runtime of the LSTM neural network filter will erode as the practicality of exchanging time for a higher accuracy inversely grows.

GitHub link: https://github.com/AndrewYinLi/lstm-neural-network-spam-filter

References

[1] Olah, C. (2018). Understanding LSTM Networks – colah's blog. [online] Colah.github.io. Available at: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[2] Hochreiter, S. and Schmidhuber, J. (1997) "Long short-term memory,“ in Neural Computation 9(8), 1997

[3] Graham, P. (2002). "A Plan for Spam.“ [online] Paulgraham.com. Available at: http://www.paulgraham.com/spam.html.

[4] Kaspersky. (2018). "Kaspersky Lab Spam and Phishing report.“ [online] Available at: https://usa.kaspersky.com/about/press-releases/2018_fifa-2018-and-bitcoin-among-2017-most-luring-topics.

[5] NVIDIA. (2018). "Accelerating AI with GPUs: A New Computing Model.“ [online] Available at: https://blogs.nvidia.com/blog/2016/01/12/accelerating-ai-artificial-intelligence-gpus.

[6] Bhowmick, A. and Hazarika, S. (2016). "Machine Learning for E-mail Spam Filtering: Review, Techniques and Trends.“ [ebook] arxiv. Available at: https://arxiv.org/pdf/1606.01042.pdf.

[7] Mikolov, T., Karafiat, M., Burget, L., Cernocky, J. and Khudanpur, S. (2010). "Recurrent neural network based language model.“ [ebook] vutbr.cz. Available at: http://www.fit.vutbr.cz/research/groups/speech/publi/2010/mikolov_interspeech2010)_IS [Accessed 19 May 2018].

[8] Zhao, S., Liu, L., Guo, M. and Yun, J. (2018). "Towards Accurate Deceptive Opinions Detection based on Word Order-preserving CNN.“ [ebook] arxiv. Available at: https://arxiv.org/pdf/1711.09181.pdf.

[9] Rusland, N., Wahid, N., Kasim, S. and Hafit, H. (2017). "Analysis of Nave Bayes Algorithm for Email Spam Filtering across Multiple Datasets.“ [ebook] IOPScience Publishing. Available at: http://iopscience.iop.org/article/10.1088/1757-899X/226/1/012091/pdf.

[10] Scikit-learn.org. (2018). 1.9. "Naive Bayes scikit-learn 0.19.1 documentation.“ [online] Available at: http://scikit-learn.org/stable/modules/naive_bayes.html [Accessed 9 May 2018].

[11] Brunton, F. (2013). "Spam: A Shadow History of the Internet. 1st ed.“ [ebook] Cambridge: MIT Press. Available at: https://books.google.com/books?id=QF7EjCRg5CIC&pg=PA136

[12] Fumera, G., Pillai, I. and Roli, F. (2018). “Spam Filtering Based On The Analysis Of Text Information Embedded Into Images.“ [ebook] MIT. Available at: http://jmlr.csail.mit.edu/papers/volume7/fumera06a/fumera06a.pdf.