Go Phish

Grant Martin
University of Texas at Austin
grantmartin2002@gmail.com

Misbah Imtiaz
University of Texas at Austin
misb2001@gmail.com

Abstract

Abstract There has been a rise in phishing email scams designed to steal sensitive information from individuals and organizations alike masked in the form of a deceptive email found in one's inbox. These faulty emails are responsible for data breaches, reputational damage, and millions in financial losses annually. In recent years, machine and deep learning techniques have been explored to help detect the difference between spam emails and distinguish them from legitimate messages. This paper explores the use of neural nets to assist in classifying emails as spam (fake) or legitimate (ham). We will use the emails gathered in the Enron data set and construct our own model, specifically a word LSTM model. We will highlight our model's architectural efficacy and optimize hyperparamaters to achieve the best results possible. We hope this study demonstrates the potential of neural networks for spam detection in emails, highlighting the benefits and challenges of using this approach in a real-world setting. The source code to our model along with the model card can be found here: https://github.com/Grantmartin2002/GoPhish

1 Background

1.1 The Implication of Phishing Scams

Phishing is a technique used to steal sensitive information such as bank account information, personal data, or company information through a fraudulent solicitation in email or on a web site, in which the perpetrator masquerades as a legitimate business or reputable person [4]. While there are a plethora of modes to distribute a phishing scam such as messaging, phone calls, websites, etc..., the main form of distribution our research focuses on is through email. Around 1.2% of emails sent daily are malicious which translates to 3 billion phishing emails sent per day [5]. This is problematic since the sheer volume of these scams creates more chances for users to fall susceptible to clicking on links and downloading files that are embedded within the emails. Additionally, falling pray to these attacks are costly. Over 2.7 billion dollars were lost at the hands of business email compromises in the US [6]. This has caused a rise in companies facilitating training program for its workers to help better identify the difference between legitimate and faulty emails. Due to its impact affecting millions of individuals and corporations alike, the problem of discerning the difference between real and fake emails has become a popular issue within the machine learning community. Our goal was to provide a solution to this issue by building our own custom machine learning model with neural nets. This would not only allow us to apply the knowledge we gained in our class, but to also create a starting point for exporting the model to use in a daily setting (maybe through a google chrome extension). The first objective was to find a data set that supplied us with enough email samples of legitimate and phishing emails.

1.2 Data Set

The data set we have chosen is known as the Enron Data set. Enron was a US-based energy company that became involved in one of the biggest financial scandals in history that led to the collapse of the

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

company in 2001 [2]. There were 600,000 emails from the company gathered and publicly released by the by the CALO Project (A Cognitive Assistant that Learns and Organizes) [3]. In 2006, a paper was published by Metsis, Androutsopoulos, and Paliouras at the third conference on email and anti-spam that took the original data and combined a mixture of faulty (spam) and legitimate (ham) emails [1]. This data set can be found here: https://www2.aueb.gr/users/ion/data/enron-spam/. From this newly infused data set, we used 16545 ham and 17171 spam pre-processed emails. The email samples itself are each stored in their own text file including the subject, body, and other general header information such as who the email is being sent to and where its coming from. We decided to go with the Enron data set for a couple of reasons. Firstly, the data set is already annotated where each email is labeled as spam or ham. Secondly, we wanted to use a data set that has a solid reputation in the context of phishing classification problems in order to benchmark our results to other models. Lastly, the data set itself stems from real world data so it provides the model a chance to train on relevant information that could be useful for real world instances of emails as opposed to using synthetic or artificially generated data.

2 Neural Net Design

2.1 Data Preprocessing

The Enron dataset of emails required several preprocessing steps to prepare the data for use in our model.

Firstly, the dataset was truncated to a maximum email length of 200 words, and the number of emails in each category (spam and ham) was balanced to avoid class imbalance during training. Specifically, emails were randomly sampled from each category to create a new dataset with an equal number of emails in each category. The resulting dataset was a list of emails and corresponding labels, where each label indicated whether the email was spam or ham.

Next, the email texts were tokenized, meaning that each word was converted to a unique index. This process was performed using a dictionary to map each unique word to an index, and each email was converted to a sequence of indices representing the words in the email. The sequences were then padded to be of equal length, which is necessary for input into the LSTM layers of the model.

Finally, the labels were converted to categorical format, where each label was assigned a unique integer value. The preprocessed dataset was split into training, validation, and test sets, with a split ratio of 60%, 20%, and 20%, respectively. The data was then loaded into PyTorch data loaders with a batch size of 64 for efficient training.

In summary, the data preprocessing steps included truncating and balancing the dataset, tokenizing and padding the email sequences, converting labels to categorical format, and splitting the dataset into training, validation, and test sets.

2.2 Model Architecture

We utilized PyTorch to implement our email classification model using a WordLSTM architecture. This model is well-suited for capturing long-term dependencies within the text. The model takes tokens (word indices) as input and consists of the following layers in sequence:

1. An embedding layer, which maps each word index in the input sequence to a high-dimensional vector representation.
2. Two LSTM layers, which process the embedded sequence and capture contextual information by maintaining a hidden state that evolves over time.
3. Two fully connected layers, which perform a nonlinear transformation of the output from the LSTM layers and produce the final classification output. The first fully connected layer uses a rectified linear unit (ReLU) activation function to introduce nonlinearity, while the final layer uses a sigmoid activation function to produce a binary classification output.

Overall, the input tokens are first embedded, then processed by the LSTM layers, and finally passed through the fully connected layers with ReLU and sigmoid activation functions to produce the output. This architecture allows the model to capture complex relationships between words in the input text and produce accurate email classifications.

The implementation of the model in PyTorch is provided below:

```python
class WordLSTM(nn.ModuleList):
    def __init__(self, input_size, hidden_dim, LSTM_layers, device):
        super(WordLSTM, self).__init__()
        self.LSTM_layers = LSTM_layers
        self.hidden_dim = hidden_dim
        self.device = device

        self.embedding = nn.Embedding(input_size, hidden_dim,
            padding_idx=0)
        self.lstm = nn.LSTM(input_size=hidden_dim,
            hidden_size=hidden_dim, num_layers=LSTM_layers,
            batch_first=True)
        self.fc1 = nn.Linear(in_features=hidden_dim, out_features=257)
        self.fc2 = nn.Linear(257, 1)
        init.xavier_uniform_(self.fc1.weight)
        init.xavier_uniform_(self.fc2.weight)

    def forward(self, x):
        h = torch.zeros((self.LSTM_layers, x.size(0),
            self.hidden_dim))
        c = torch.zeros((self.LSTM_layers, x.size(0),
            self.hidden_dim))
        h = h.to(self.device)
        c = c.to(self.device)

        torch.nn.init.xavier_normal_(h)
        torch.nn.init.xavier_normal_(c)

        out = self.embedding(x)
        out, (hidden, cell) = self.lstm(out, (h,c))
        out = self.fc1(out[:,-1,:])
        out = torch.relu_(out)
        out = self.fc2(out)
        out = torch.sigmoid(out)
        return out.squeeze()
```

2.3 Learning and Hyperparamters

The email classification model was trained using the Adam optimizer with a learning rate of 0.001 and the binary cross-entropy loss function was used as the objective function. To improve the model's performance, we incorporated a learning rate scheduler with a patience of 2 and a factor of 0.5, meaning it halves the learning rate when the validation loss stops improving for 2 epochs.

The hyperparameters used in the model include a hidden size of 128 an LSTM layer count of 2. The model was trained for 20 epochs on the training set with a batch size of 64, and we monitored the validation loss at the end of each epoch. To track the model's performance, we recorded the training and validation losses for each epoch and printed the average train and validation loss.

3 Conclusions

3.1 Model Results

After performing the training on 60 percent of the data set, we accumulated the results from 20 percent of the data set and measured validation with the other 20 percent.

Looking at figure 1, we can see how our architectural decisions have impacted the way the model is learning. For example, since we used a ReduceLROnPlateau for our learning rate scheduler, we can see the learning rate start to reduce when the validation loss stagnates. This reduction in the learning rate allows the optimizer to take smaller steps and avoid overshooting the optimal solution. After running 20 epochs, we were able to visualize our results in figure 2 and 3.

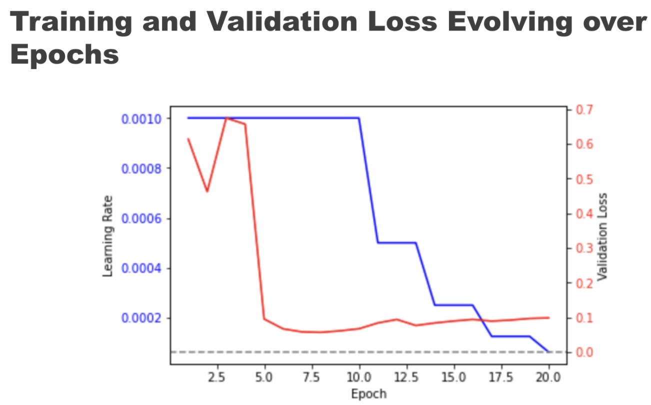

Figure 1: Displaying how our validation loss and learning rate evolved after every epoch

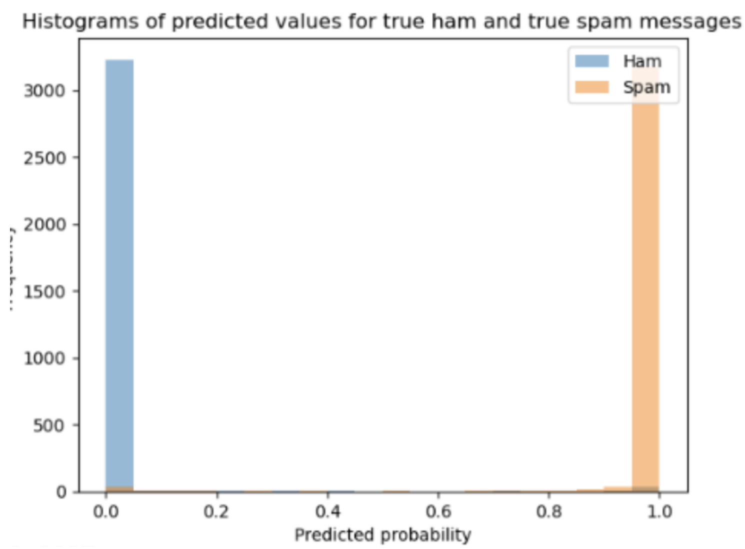

Figure 2: Histogram of predicted values for spam and ham messages

Our model achieved an accuracy of 98.37% with a precision and recall metric of 98.51% and 98.22% respectively. Looking at figure 2, there is only a sliver of miscoloration on each of the bars representing spam or ham, indicating there was not many false positive or false negatives that our model predicted. Additionally, this is explicitly shown within the figure 3, as the confusion matrix shows the false negative and positive sections are considerably lower then the true negative and positive sections. This led our F1 score to be high as well at 98.37%. Furthermore, we had an AOC-ROC score of 99.63%. The high ROC-AUC score indicates that the model is able to make very accurate predictions and is performing very well overall. These high results and metrics can be caused by a multitude of factors. Firstly, our architecture of our model was created in a way to detect the general themes of spam and ham emails. For example, the LSTM layer is effective in capturing long-term dependencies in the sequence of text data, which can be important for identifying patterns. In addition, our embedded layer was important for catching the semantic meaning of the words in the email, which assisted in identifying specific words and phrases that were indicative of spam or ham. Secondly, the data set itself were only emails in the context of what employees send amongst each other at Enron. If we had used emails from a variety of sources our accuracy may have been lower. There were some more challenges and difficulties outlined in the next section.

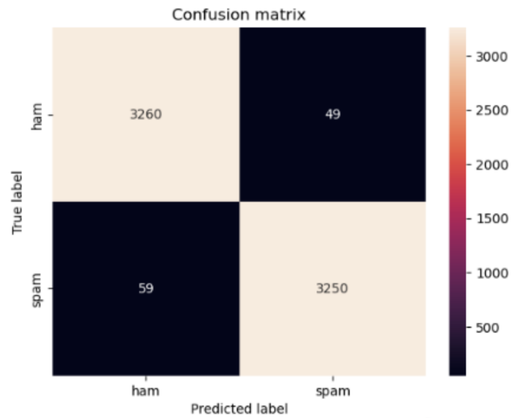

Figure 3: Confusion matrix of the predicted values

3.2 Model Challenges and our Solutions

When initially training the model, we realized that we were suffering from vanishing gradients as our model stopped learning after the third epoch and identified all emails as spam. At first, we thought this was an architectural issue, but later, we realized it was how our email data was fed into the model. For example, since we were processing the entire thousand-word emails at once, the long input sequences prevented our model from properly learning the patterns as the gradients were vanishing as they were back propagated through time. To combat this, we truncated the emails to the first 200 words which helped improve the accuracy immensely. In the future, we would like to choose 200 words randomly within the email to reduce bias. Secondly, using LSTMs was computationally expensive. Running a single epoch would have taken hours which would prevent us from having enough time to find the optimal hyperparameters. To tackle this issue, we used the CUDA toolkit which allowed us to move the stress of training the model off of the CPU and onto NVDIA's GPU on our local machines. This drastically reduced training times and helped us find the parameters that best-improved accuracy.

3.3 What We Have Learned

Throughout the process of creating the neural net model, we gained several valuable insights. First, we learned about different loss functions and how they can be used to measure the efficacy of a model. For example, we experimented with the binary cross-entropy loss function and found it most effective for our classification problem. Additionally, we used a learning rate scheduler to help our model converge to a better accuracy over time. Second, we gained a deeper understanding of hyperparameters and how tuning them can affect the accuracy of a model. We experimented with various hyperparameters, such as learning rate, batch size, and number of layers. Third, we learned about LSTM cells and embedded layers within a neural net model architecture for modeling sequences of data such as text or speech. Looking forward, we plan to continue exploring new techniques and approaches to NLP problems and building on the foundation we established during this project.

3.4 How We Used ChatGPT

We used ChatGPT just as we had used in our homework 4 assignment. This involved asking the model conceptual questions. For example, when having issues with what loss function to use, how to speed our training of the model, how to do the dimensions change after a certain layer, why our model wasn't learning etc..., ChatGPT helped us better understand larger conceptual ideas and problems and offered a sense of direction for us to look into. Secondly, we used ChatGPT when facing syntax and bug issues. Instead of using stack overflow and going through several pages to find an answer to why our tensor was throwing an error, we turned to ChatGPT to get an immediate answer.

Contributions

We worked together and pair programmed on tasks but had general ownership on different sections.

Misbah Imtiaz

- Load Enron Data set
- Truncate and Balance the Data
- Convert to list of emails and corresponding labels and shuffle
- Define data loaders
- Tokenize and Pad Sequences

Grant Martin

- Define Model
- Intialize Model
- Train Model
- Evaluate Model

References

[1] Androutsopoulos, I., Koutsias, J., Chandrinos, K. V., Paliouras, G., Spyropoulos, C. D. (2006). Spam filtering with naive Bayes - Which naive Bayes? Proceedings of the 3rd Conference on Email and Anti-Spam (CEAS), Mountain View, CA, USA, 26-27 July 2006, 17-28.

[2] Cohen, W. (2015, May 8). Enron Email Dataset. Enron email dataset. Retrieved April 25, 2023, from https://www.cs.cmu.edu/ enron/

[3] Enron Corp Cohen, W. W. (2015) Enron Email Dataset. United States Federal Energy Regulatory Commissioniler, comp [Philadelphia, PA: William W. Cohen, MLD, CMU] [Software, E-Resource] Retrieved from the Library of Congress, https://www.loc.gov/item/2018487913/.

[4] Editor, C. S. R. C. C. (n.d.). Phishing - glossary: CSRC. CSRC Content Editor. Retrieved April 24, 2023, from https://csrc.nist.gov/glossary/term/phishing

[5] James, N. (2023, April 5). Phishing statistics and DMARC. EasyDMARC. Retrieved April 25, 2023, from https://easydmarc.com/blog/phishing-statistics-easydmarc-report-january-june-2022/#: :text=Globally

[6] The latest phishing statistics (updated April 2023): Aag it support. AAG IT Services. (2023, April 13). Retrieved April 25, 2023, from https://aag-it.com/the-latest-phishing-statistics/#: :text=The