SMS Spam Style Transfer

Karen Chung, Saketh Gabbita, Philip Li, Eric Wang, Helen Yang

Abstract

Spam communication has been a problem since the inception of the internet a few decades ago. More recently, researchers have found that users are more exposed to attacks through Short Message Service than through email. While there has been a lot of research in NLP and more general spam classification tasks involving textual reconstruction and anomaly detection among others, very little has been done to intentionally reconstruct spam-style messages using style transfer. In this project, we attempted to gain a better understanding of the underlying characteristics of spam messages versus their "standard" message counterparts. In doing so, we made use of the NUS SMS Corpus consisting of several thousand text messages labeled as either spam or not. We then constructed a model involving several different classical architectures/techniques in NLP literature, primarily an autoencoder implementation with an attention mechanism, trained on the aforementioned corpus. With this in place, we analyzed the output message from our model given an input text message in terms of its likelihood to be an actual spam message.

1 Introduction

Over the past three decades, the internet has surged in popularity and served as a powerful impetus for globalization and human connection. Despite its many benefits, this framework has enabled adversaries to take advantage of users who are not well versed in cybersecurity protocols, frequently attacking their critical systems and gaining access to sensitive information like credit card numbers and personal information (social security, birthday, location, etc.). While electronic-mail (email) comprises most of this type of spam communication, there has also been a steady increase in the amount of spam being delivered via Short Message Service (SMS). Some researchers even argue that SMS spam is actually more dangerous than email spam because of a higher proclivity to open an SMS message than an email (M2 Computing). However, many sophisticated methods have been developed to block spam, but mostly as they relate to spam filters on websites (primarily through email). There have been similar endeavors as it relates, to SMS, but these exist more sparsely in NLP literature pertaining to spam classification tasks. While these models are not always correct, they are very effective in filtering out spam based on content, headers, language, and a number of other rules governed by the niche or non-standard formatting of the overwhelming majority of spam communication today.

In response to this context, our project endeavors to alter typical communication, or a standard message, (via SMS primarily) to achieve a spam-like message that is produced by many programmed bots. A deeper understanding into this relationship has a tremendous amount of potential in making spam filters even more robust in a the SMS setting which hasn't been explored nearly to the extent of its email counterpart. Specifically, we hope to discover ways in which spam creators are currently attempting to propagate spam undetected by manipulating text and attempt to predict the ways in which they may engineer their spam so it is less likely to be detected. By analyzing the results of our model and the parameters/features that it has detected, we gain very valuable insight into the complex relationship between SMS spam creators and the filers that attempt to detect it as they continue to evolve.

2 Related Work

As mentioned earlier, there has been a lot of work done in NLP that relates, at least to some extent, either to the motivation of our project or the specific methods that we used to complete it.

A significant paper in the literature pertaining to style transfer authored by Jin et al. serves as a primary motivator in some of our stylistic, architectural choices. While there have been numerous linguistic approaches to text style transfer, the success of deep learning in the past few years has given rise to a plethora of neural network methods. Seq2seq has often been used in the case of parallel data, while other approaches such as disentangling text into its content and attribute, prototype editing, and pseudo-parallel corpus construction have further increased the robustness of text style transfer, especially with the advent of transformers. Their work discusses the state of the art techniques on text style transfer as well as different modes of evaluation for text style transfer. Some methods include autoencoders, Generative Adversial Networks (GANs), and Variational Auto-Encoders (VAE). They provide many loss functions such as Bag of Words Overlap, Adversial BoW Overlap, and Cycle Reconstruction. Of particular note is their work with autoencoders which motivates a solution for us in terms of learning the most pertinent features of spam communication by first encoding given data and attempting to regenerate the input from the encoding.

Another influential paper in the field that we drew inspiration from in developing our model architectures was one by Barak Oshri which essentially investigated the role of autoencoders in the textual reconstruction task. The main idea behind an autoencoder is that features can be extracted from some the compressed form of some input (in this case a sentence). The paper showed that it was feasible to develop an autoencoder scheme that successfully discerns useful English syntax rules through its reconstructive features. Furthermore, the structure that Oshri deployed to achieve this was a recurrent neural network (RNN), and the paper recommended using gated rectifier units (GRUs) as introduced nonlinearity to the model. In essence, this further shows that the idea of reconstructing an input using an autoencoder in a textual setting (in our case for spam/non-spam messages) is not only feasible, but also successful.

One final paper that relates to our project more broadly is one by Eleazar Eskin in which he goes over the process of detecting errors within a corpus using anomaly detection. He goes about this task by first computing a probability distribution over the input of the entire corpus. He then applies a statistical test which identifies which elements are anomalies, usually elements with very low statistical likelihood (calculations are actually done using log-likelihood since this is less time expensive and still the same problem since the log function is monotonically increasing over the relevant domain). These elements are marked as errors, and the results of the model showed that he was able to have achieve a 44 percent accuracy of detecting corpus anomalies (as calculated by the quanitity corpus errors/(corpus errors + system errors)) once reviewed by NLP researchers. Though this paper addresses a broader problem than the one we are tackling, it is useful to consider this idea in the context of spam style transfer because the anomaly detection framework that he uses provides some insight into the method that we may be able to use to differentiate between spam and non-spam messages. Furthermore, using anomaly detection/testing can help illuminate any errors or elucidate some of the underlying features of the messages in our corpus.

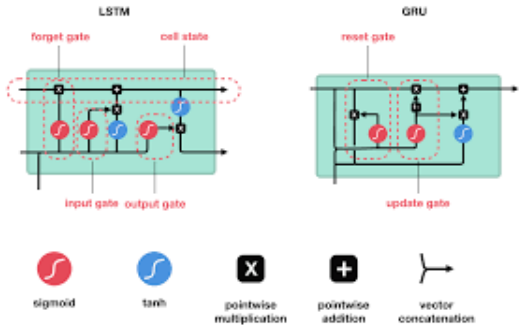

Figure 1: A diagram of LSTM and GRU architecture.

3 Model Architecture

Our SMS model is based on an autoencoder architecture. Specifically, the model consists of two RNNs: an encoder LSTM and a decoder LSTM. Over a single pass through the autoencoder, we expect to reconstruct the input sentence.

We use a many-to-one encoding structure. The encoder reads in the input sequence of words, and after the last input token has been processed, the state of the LSTM is essentially a representation of the whole sequence. An LSTM is a simple structure that features an input gate, a forget gate, and an output gate - together, these mechanisms allow the LSTM to encode information across long sequences and control the flow of information across tokens within a sequence. Due to this structure of

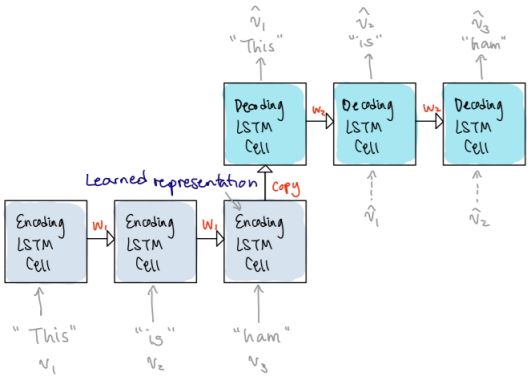

Figure 2: Illustrates our baseline autoencoder model architectures that are implemented in our experiments.

each LSTM cell, we expect to be able to recover long-range characteristics of input ham messages; in other words, we aim to generate sensible outputs not only at the individual word level, but also at the overall structural level of legitimate ham SMS messages.

We generate that output (which should be a reconstruction of the original ham message) using our decoder. Given the vectorized representation of an input sequence from the final state of our encoder, the decoder sequentially generates a reconstructed sentence. We use a one-to-many decoder; we "unroll" the RNN over the length of the sequence, ultimately retrieving a final hidden state that represents the final state for each element in the sequence of words. Intuitively, this decoder model must be trained in its ability to (1) output an appropriate word based on the current hidden sequence and (2) evolve to the next hidden state given the current one.

As seen in Figure 2, the baseline model structure (LSTM encoder-LSTM decoder) is appropriate for assessing the general fit of our autoencoder approach to the SMS style-transfer task. In addition, we identify several architectural features that, when integrated into the model, may improve performance. Concretely, we construct models with (1) stacked LSTM layers for the encoder, (2) bidirectional GRUs (instead of LSTMs) for the encoder and (3) LSTM model with attention. We hypothesize that additional layers (i.e. neural network complexity), bidirectionality (i.e. learning forward *and* backward relationships between words), and attention (i.e. attending to every hidden state from encoding step) may help the reconstruction task.

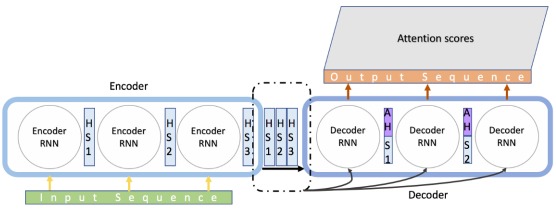

Figure 3: Another overview of our model architecture with attention added.

Figure 4: Another example of our dataset.

4 Corpus

We used the following SMS text message dataset to train our autoencoder: https://www.dt.fee.unicamp.br/~tiago/smsspamcollection/. We download the entire dataset, consisting of 5,574 English, real text messages, which were labelled as either legitimate (ham) or spam. These messages came from a variety of sources, including 3,375 random messages from the NUS SMS Corpus (NSC), another dataset consisting of about 10,000 legitimate messages collected by the Department of Computer Science at the National University of Singapore. Many spam message examples come from Grumbletext, an online forum for users to make public complaints about SMS spam messages they receive. Although the dataset is relatively small, we felt that it was sufficient for the purposes of our project given the task of style transfer and the limited scope of our computing power.

The dataset is formatted in plain text, where each line consists of the class label (spam or ham) followed by a tab and the raw text of the message. In order to preprocess our dataset for the purposes of style transfer, we decided to create two distinct datasets, one with only "ham" messages and the other with only spam messages. Within the dataset, there are 4,827 occurrences of non-spam (ham) messages, and 747 occurrences of spam messages. That is, our dataset is about 86.6% ham and 13.4% spam messages, or a majority of ham messages.

We shuffled the dataset randomly and split it into

```
ham   What you doing?how are you?
ham   Ok lar... Joking wif u oni...
ham   dun say so early hor... U c already then say...
ham   MY NO. IN LUTON 0125698789 RING ME IF UR AROUND! H*
ham   Siva is in hostel aha:-.
ham   Cos i was out shopping wif darren jus now n i called him 2 ask wat
present he wan lor. Then he started guessing who i was wif n he finally
guessed darren lor.
spam  FreeMsg: Txt: CALL to No: 86888 & claim your reward of 3 hours talk
time to use from your phone now! ubscribe6GBP/ mnth inc 3hrs 16 stop?txtStop
spam  Sunshine Quiz! Win a super Sony DVD recorder if you canname the
capital of Australia? Text MQUIZ to 82277. B
spam  URGENT! Your Mobile No 07808726822 was awarded a L2,000 Bonus Caller
Prize on 02/09/03! This is our 2nd attempt to contact YOU! Call 0871-872-
9758 BOX95QU
```

Figure 5: Example messages from our dataset.

```
Here are some vocabs from the spam SMS corpus:
['comes', 'Freeentry', 'UKP', 'lionp', 'Ever', 'bears'
There are 3645 unique vocabs in the spam corpus


Here are some vocabs from the ham SMS corpus:
['EY', 'resort', 'pix', 'sweater', 'TELL', 'fumbling',
There are 8434 unique vocabs in the ham corpus
```

Figure 6: Example words from our dataset.

two parts: 75% for training and 25% for testing, which include 3620 and 1207 samples respectively. We have two textual components that we use for our classification task: the label (spam or ham) and the raw text message. These components are extracted from the raw text file and processed to be fed into our model. Further analysis of our dataset reveals that it is well-built for our task; the ham and spam messages come in variable lengths and configurations, but share the same average length and relative distribution of lengths.

5 Implementation

In order to build and train our network, we processed our data in Python and used PyTorch modules to build the layers described for our autoencoder structure.

To process the dataset, we split each line in the raw text file and distinguished between labels and messages using the tab delimiter that separate the labels from the messages. We then put each label and each message instance into a Pandas Dataframe, which is then be loaded into our model for later use. We also queried from our dataset in order to examine the internal data to ensure that it was what we expected and analyzed the distribution of the length/sizes of the individual messages.

We create a class called SMSDataset, which stores all the entries of the messages. We use indices such as SOS index to mark the start of the

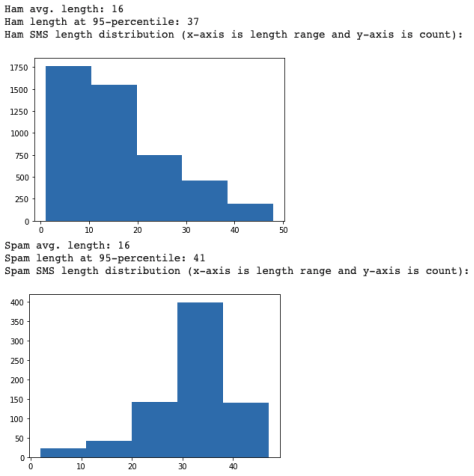

Figure 7: The distribution of words lengths in our dataset

```
#Dataloader class for gpu efficiency
from torch.utils.data import Dataset, DataLoader

# These IDs are reserved.
PAD_INDEX = 0
UNK_INDEX = 1
SOS_INDEX = 2
EOS_INDEX = 3

class SMSDataset(Dataset):
  def __init__(self, all_sentences, all_vocabs, sampling=1.):
    # 0) constants
    self.max_sms_seq_length = MAX_SENT_LENGTH_PLUS_SOS_EOS  # determine based on sentence lengths distribution

    # 1) sentence data
    self.all_sentences = all_sentences[:int(len(all_sentences) * sampling)]

    # 2) SMS vocabulary
    self.all_vocabs = all_vocabs

    # id-match vocabulary
    self.sms_v2id = {v : i for i, v in enumerate(all_vocabs)}
    self.sms_id2v = {val : key for key, val in self.sms_v2id.items()}

  def __len__(self):
    return len(self.spam_sentences) + len(self.ham_sentences)

  def __getitem__(self, index):
    sms_sent = self.all_sentences[index]
    sms_len = len(sms_sent) + 2   # add <s> and </s> to each sentence
    sms_id = []
    for w in sms_sent:
      if w not in self.all_vocabs:
        w = '<unk>'
      sms_id.append(self.sms_v2id[w])
    sms_id = ([SOS_INDEX] + sms_id + [EOS_INDEX] + [PAD_INDEX] *
              (self.max_sms_seq_length - sms_len))


    # we return the same thing twice because autoencoder
    return torch.tensor(sms_id), sms_len, torch.tensor(sms_id), sms_len
```

Figure 8: The implementation of our dataset module.

sentence and the end of the sentence, and also PAD-index for the end of the sentence all the way to the max-length. We also mimic what we did in class, which was label "unk" using the unk index for words that are not in the dictionary that are found in the sentences. One thing we might do in the future is to detect misspellings or slang/informal ways of spelling a word that are common in spam messages and mark these. This will be more specific than simply marking everything that cannot be found in the dictionary as "unk".

To implement our model architecture, we utilize two of our own defined modules – an EncoderRNN model and a DecoderRNN model, both of which inherit from PyTorch's nn.Module class. For both the encoder and decoder, we define forward functions that feed the input through a GRU and intialized hidden states. Both the encoder and the decoder are then combined into a single model, which first feeds the vectorized messages through

```python
class BahdanauAttention(nn.Module):
    """Implements Bahdanau (MLP) attention"""

    def __init__(self, hidden_size, key_size=None, query_size=None):
        super(BahdanauAttention, self).__init__()

        # We assume a bi-directional encoder so key_size is 2*hidden_size
        key_size = 2 * hidden_size if key_size is None else key_size
        query_size = hidden_size if query_size is None else query_size

        self.key_layer = nn.Linear(key_size, hidden_size, bias=False)
        self.query_layer = nn.Linear(query_size, hidden_size, bias=False)
        self.energy_layer = nn.Linear(hidden_size, 1, bias=False)

        # to store attention scores
        self.alphas = None

    def forward(self, query=None, proj_key=None, value=None):
        # assert mask is not None, "mask is required"

        # We first project the query (the decoder state).
        # The projected keys (the encoder states) were already pre-computated.
        query = self.query_layer(query)

        # Calculate scores.
        scores = self.energy_layer(torch.tanh(query + proj_key))
        scores = scores.squeeze(2).unsqueeze(1)

        # Turn scores to probabilities.
        alphas = F.softmax(scores, dim=-1)
        self.alphas = alphas

        # The context vector is the weighted sum of the values.
        context = torch.bmm(alphas, value)

        # context shape: [B, 1, 2D], alphas shape: [B, 1, M]
        return context, alphas
```

Figure 9: Implementation of the Bahdanau Attention module.

the encoder, and then feeds the encoded messages through the decoder to recreate the messages in the style of a spam message. Furthermore, we use PyTorch's DataLoader module with a batch size of 128 to load our messages and labels and provide an iterable over our dataset for ease of use with our model.

We also decided to employ a Bahdanau (MLP) attention module (Figure 9), which we believed would be helpful in learning seq2seq attention associations in our data sequences. The module essentially replaces a fixed-length vector with a variable length one and allows target words produced by the decoder to focus on relevant sections of the input message, thus enhancing the quality of our style transfer process. By doing so, it will improve the performance on longer sentences, and in our case, it will be of a great use as many spam or ham messages reach 45 words or longer.

To implement attention, we have three layers consisting of key, query, and energy. We store the attention scores in self.alphas, which refers to the weights of previous tokens. In the forward function, which is called every iteration, the attention mechanism, takes the previous hidden state of the decoder and the list of encoded words, and uses them to generate unnormalized score values that indicate how related the elements of the input sequence align with the current output, through the energy layer. The scores are then normalized by passing them

```python
# Hyperparameters for contructing the encoder-decoder model.
embed_size = 256  # Each word will be represented as a `embed_size`-dim vector.
hidden_size = 512  # RNN hidden size.
dropout = 0

attention = BahdanauAttention(hidden_size)

# Construct the autoencoder
pure_seq2seq = EncoderDecoder(
  encoder=Encoder(embed_size, hidden_size, dropout=dropout),
  decoder=Decoder(embed_size, hidden_size, dropout=dropout, attention=attention),
  sms_embed=nn.Embedding(len(spam_vocab_set), embed_size),
  generator=Generator(hidden_size, len(spam_vocab_set))).to(device)

# Start training. The returned `dev_ppls` is a list of dev perplexity for each epoch.
pure_dev_ppls = train(pure_seq2seq, num_epochs=20, learning_rate=1e-3,
                      train_data_loader=spam_train_data_loader,
                      val_data_loader=spam_test_data_loader,
                      task="seq2seq", print_every=100)

plot_perplexity(pure_dev_ppls)
```

Figure 10: Implementation for training our model.

through a softmax function to generate the weights. Following the softmax normalization, all of the weight values will lie in the interval [0, 1] and will add up to 1. Every iteration, there is also a context vector that is calculated based on the score and is used to calculate the hidden layer in each step of the decoder.

For training our model, we utilize PyTorch's Adam optimizer and cross-entropy loss. We then iteratively apply the backpropagation algorithm to update model weights and improve the fit of our model output to the provided labels from the dataset. At each iteration, we evaluate the loss and accuracy of our model.

6 Experiments and Results

Using a hidden size of 512 for the RNN module, we were able to achieve a final validation perplexity value of 6.334282 at epoch 8 of training (Figure 11), which is a relatively low value for style transfer tasks and suggests that the test set was well represented. Although we initially found it difficult to successfully train our model and achieve acceptable results, we eventually found that an embed size of 256, a hidden size of 512, 20 epochs, and a learning rate of 1e-3 produced a relatively low complexity score that we were satisfied with. Further inspection of the output messages produced by our model reveals that the text appears very "spammy" and would likely be detected by modern spam detection filters.

Although the spam classification model is by no means a ground truth, it provides valuable insight into how our adversarial model might perform in practice. Using an annotation team would have further allowed us to obtain an accurate measure of how our model is performing, but given the relatively short time frame we decided to just manually inspect the results ourselves.

Given the relatively small size of the dataset,

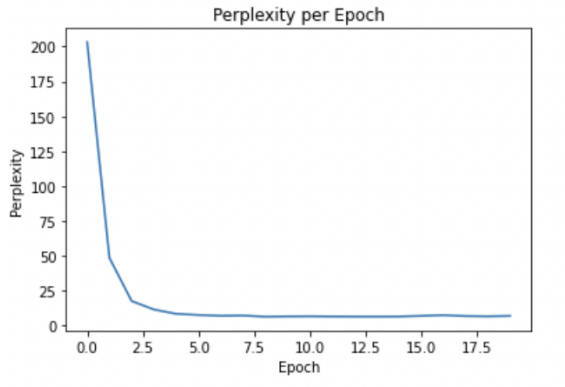

Figure 11: Perplexity per epoch of training.

we also had the chance to perform rigorous cross-validation and reshuffled our original dataset into various combinations of training/testing datasets. Although it was difficult to arrive quickly at optimal training conditions for our model, between training sessions we found that we were able to tune hyperparameters and achieve results that we were content with via manual inspection.

Further examining the perplexity, we note that the validation perplexity measures how correlated a generated spam message is to an actual spam message. Essentially, the lower the validation perplexity, the more it resembles an actual spam message. We see in Figure 11 that the training is effective, as there is no overfitting or underfitting in the later epochs. We also note that after 2.5 epochs, the model already reaches its equilibrium point of around 8 validation perplexity. The validation perplexity starts at 500 in the first epoch, 100 in the second epoch, and in the third epoch, reaches 8, the equilibrium point. This means that the autoencoder is very powerful and potentially, it can be used to do even more than the current task at hand. Furthermore, we also noted that the loss was consistently decreasing - it started from 300 and gradually decreased to 0.064891.

In our code for the training section, we have that the learning rate is 0.001 for the raining and in the autoencoder, we use a generator with dimensions hidden-size and an embedding layer as well. Overall, the graph proves that our training is effective, as throughout merely 20 epochs, the validation perplexity settles at a desirable value of around 6. Further, no overfitting occurs even though the model already fit the data well with relatively few epochs.

Another quantitative evaluation we decided to calculate was the BLEU score of our model. BLEU score measures the similarity of the machine-translated text to a set of high quality reference translations. This score will tell us how similar our model outputs are to actual examples of spam messages, giving us a measure of model quality outside of just manually inspecting the outputs of our model. Using our attention model, we were able to get a BLEU score of 4.231. Since we're not trying to get our outputs to match any labels in particular and are just evaluating a "similarity" to spam messages (thus we don't need exact matches), we believe that this BLEU score is sufficient.

	Perplexity	BLEU Score
No attention	161.564390	0.125
With attention	6.334282	4.231

Table 1: Table displayed evaluation metrics with and without attention.

Figure 12: Example spam messages generated by our model.

We also note that adding attention to our decoder greatly improved the performance of the model. This result makes sense intuitively because adding attention allowed our model to dynamically highlight the important, relevant features of our input data. This leads to the analysis that perhaps there are certain key words or phrases found in spam messages that are not found in regular ham messages that lead a message to be seen as spam versus ham. In Table 1, we compare the results of our model with and without attention mechanism.

Figure 12 displays a few examples of the outputs from our SMS transfer model. The first two rows of sentences are the ham messages that we take as input. We input the ham message into our autoencoder and by doing so, we can extract features in their encoded form. The encoder will take in the input ham message and convert them into a decoded matrix, and the decoder then converts this message back to words/sentences. We are able to compare the encoded data between our ham and spam messages, which is an extremely useful way to analyze these two types of messages and find distinguishing features. Our decoder returns the exact

same sentence, which means that our autoencoder works effectively.

In Figure 12, we see that the spam messages greatly resemble spam messages in real life, as shown by the numerous spelling and grammatical errors. Many of the spam messages generated also mention money (in example 2, FREE is in all caps) and gives contact information or an address of where to send money to. In the first example, the message is clearly from a dating app that is trying to lure the recipient to contact a stranger by giving them their number. Clearly, this message is not from someone the recipient knows and is also not from a legal dating site, which would not send such information through SMS. Therefore, it appears that using an attention mechanism and autoencoder model, we were able to reliably generate real spam-like messages; this lends credence to the hypothesis that spam messages contain some inherent linguistic structure that makes them easy to distinguish as "spam" over "ham".

7 Future Work

While we able to make significant progress toward our initial goal, there still exist many ways for us to achieve the task we outlined more completely. Firstly, a necessity in being able to understand the relation between spam and non-spam messaging is extracting the relevant or distinguishing features from the autoencoder components of the model. This involves perhaps looking into the final parameters of the trained model and observing how they respond to different kinds of input, and thus how variable our output is with respect to the parameters. Furthermore, interpretability is a growing necessity for many of the sophisticated models that are being developed in artificial intelligence endeavors more broadly. Making use of the some of the visualization tools that exist can also give us stronger insight into the characteristics that are most pertinent to non-spam versus spam messaging, and how our model translates one to the other. Another method by which we can improve our model is initially using some form of k-means-clustering or principal component analysis of the different classes of messages (and their encodings specifically) to identify dimensions which yield the most information.

Another area of improvement for our project is with respect to evaluation criteria. Currently, we have a few quantitative metrics that inform how spam-like the outputs of our style transfer model are. However, beyond these measures and our own qualitative analysis of the output, there are definitely ways to evaluate our model more comprehensively. One method that we brainstormed initially was implementing the current state-of-the-art SMS spam detector (discrimination tool) that exists in the NLP literature today and feed in the outputs of our style transfer model to it. This would provide better context for the efficacy of our model.

A natural extension of our project is taking the reverse direction of our outline and implementing style transfer to convert spam messages into non-spam style messages. This objective is very difficult and potentially impossible given the limited size of our dataset, as we only have 5,500 spam and ham messages, as ham messages are more nuanced and have a lot more variability than spam messages. A more comprehensive corpus either open on the internet or compiled by us would aid in this initiative. Work towards this objective would be a leap for SMS-spam classification endeavors as well, since this would yield more comprehensive datasets. This would result in a tougher detection task for spam-detection models and those that incorporate this data would be more robust in general.

8 Conclusion

Overall, our project hoped to make progress towards improving the robustness of existing spam filters, specifically in the context of SMS spam, by designing an adversarial system that will attempt to deceive them. After implementing and training our own model on an existing dataset to accomplish this via an autoencoder-attention model, we've shown that a style transfer approach to this task is not only feasible, but also effective. In it's current state, our model takes in input of ham messages and generates corresponding spam-style messages, meeting the objective we set out to complete. Despite this, there is still more work to be done to thoroughly improve the robustness of current SMS spam filters. Chief among potential areas for further exploration is interpretability of our model via feature extraction and visualization tools. Other areas of improvement are the evaluation of our spam messages against current state-of-the-art SMS spam filters and implementing style transfer to take spam messages to ham-style messages to make datasets comprehensive. With that, we hope that our model can provide valuable insights into the characteristic language qualities of spam messages and aid in

the enhancement of current SMS spam detection filters.

References

Almeida, T. A., Hidalgo, J. M. G., Yamakami, A. (2011). Contributions to the Study of SMS Spam Filtering: New Collection and Results. Proceedings of the 11th ACM Symposium on Document Engineering, 259–262. Presented at the Mountain View, California, USA. doi:10.1145/2034691.2034742

Eskin, Eleazar. "Detecting Errors within a Corpus using Anomaly Detection." ANLP (2000).

Jin, D., Jin, Z., Hu, Z., Vechtomova, O., Mihalcea, R. (2021). Deep Learning for Text Style Transfer: A Survey. arXiv [cs.CL]. Opgehaal van http://arxiv.org/abs/2011.00416 Johnson, Joseph. "Spam Statistics: Spam E-Mail Traffic Share 2019." Statista, 20 July 2021, https://www.statista.com/statistics/420391/spam-email-traffic-share/.

Lee, D., Tian, Z., Xue, L., Zhang, N. L. (2021). Enhancing Content Preservation in Text Style Transfer Using Reverse Attention and Conditional Layer Normalization. arXiv [cs.CL]. Opgehaal van http://arxiv.org/abs/2108.00449

Mir, R., Felbo, B., Obradovich, N., Rahwan, I. (2019, Junie). Evaluating Style Transfer for Text. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 495–504. doi:10.18653/v1/N19-1049

Oshri, Barak. "There and Back Again : Autoencoders for Textual Reconstruction." (2015).

"SMS Spam More Dangerous than Email Spam." M2 Computing, 11 Sept. 2013, https://www.m2computing.co.uk/sms-spam-more-dangerous-than-email-spam/#:~:text=Malicious%20text%20messages%20are%20posing,every%20day%20in%20the%20UK.