Automating Fake Online Reviews:
How can automated reviews be improved to escape detection?

Kalvin Kao
UC Berkeley School of Information
kalvin.kao@ischool.berkeley.edu

Abstract

As the problem of fake online product reviews grows, researchers have begun investigating the potential for automating review writing, which could be cheaper, more scalable, and even more difficult to detect than the current opinion spamming ('crowdturfing') methods, which involve paying workers to write false reviews in online crowdsourcing campaigns. Previous studies have demonstrated that recurrent neural networks (specifically long-short term memory models, or LSTMs) can generate realistic reviews for automating opinion spam attacks, and can also detect machine-generated reviews for defending against such attacks. The following paper examines how one such model is built and trained, and explores its potential use in an experimental generative adversarial network (GAN) architecture for creating reviews that are more difficult to detect. A selection of experiments and analyses is featured, and leads to the identification of meaningful future areas of research in this problem area.

Introduction

When searching for products or services online, internet users are often able to read the opinions of other users and this informs their decision about which products or services to use. For example, Yelp.com features details of restaurants in most cities, including their hours, average user rating, and accepted payment methods, but its primary offering is a large space for users to freely write about their experiences at a restaurant in an unstructured manner, for the purpose of informing future customers.

While such product or service reviews may be helpful for a potential customer, there exists a problem in which a business may artificially create positive reviews for itself or negative reviews for a competing business (called false/fake reviews or deceptive opinion spam), and this is known to be the case for many major e-commerce and review websites such as Amazon, TripAdvisor, and Yelp [1,2]. Studying the potential threats of fake reviews and also the potential defenses against them is thus highly important to improving the quality of e-commerce and online information.

In "Automated Crowdturfing Attacks and Defenses in Online Review Systems", Zhao et. al. feature a character-level LSTM language model capable of producing realistic restaurant review text (an 'attack' model), for the purpose of exploring the potential threat of and defense against machine-generated opinion spam [3]. The following study attempts to build on this work by examining important factors in building such a model, and by extending it into a generative adversarial network (GAN). Specifically, this study seeks to answer the question "How can automated reviews be improved to escape detection?"

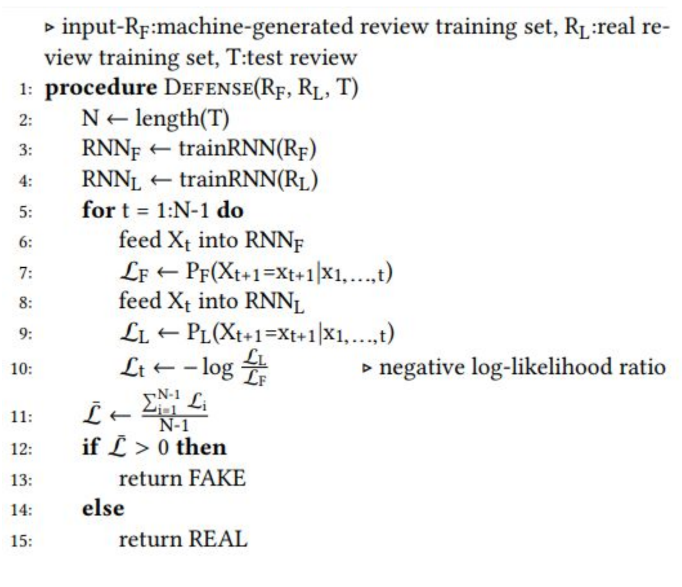
A GAN consists of a 'generator' and a 'discriminator', both being neural network models. During training, the generator attempts to produce new examples that the discriminator cannot distinguish from real training examples, and the discriminator attempts to discern real examples from those created by the generator. In the context of false reviews, such joint training of a generator and a discriminator may result in even more realistic machine-generated reviews, as their losses push against each other during training [4].

Background

Current methods for detecting fake reviews involve using such review metadata as time of posting (for example, studies have shown that fake review campaigns tend to result in ‘bursts’ of reviews-- a high number of review posts occurring in a short time-frame) and some linguistic features [5]. The attack and detection methods surrounding these features are described as a game of cat and mouse between businesses/attackers and service providers, and focusing on the linguistic features of review text may be a more reliable defense method going forward [3]. This is challenging however, because fake review text is currently written by real users to seem genuine [6].

Zhao et. al. were able to match this level of realism by using a large dataset of 1.8 million crawled Yelp reviews to train a character-level language model, which could then generate very realistic review text automatically. Named the ‘attack’ model, it is a two-layer LSTM, with each layer containing 1024 hidden units. To train the model, input strings were split into batches of size 256, training loss was computed using cross-entropy, weights were updated using Adam optimization, and training was run for 20 epochs at a learning rate of 2e-3. Additionally, review text was pre-processed by removing all extra white spaces and non-ASCII characters, and by adding the “<SOR>” (start of review) and “<EOR>” (end of review) tokens to the beginning and end, respectively, of each review. Post-processing was also done for samples generated by the attack model to provide some customization to the final artificial reviews. The training dataset itself included 617k 5-star reviews from 27k restaurants (which amounted to 57M words and 304M characters).

Zhao et. al. also developed a ‘defense’ model designed to detect reviews generated in this manner. The defense model consists of two additional LSTMs-- one trained on real reviews, and one trained on the machine-generated reviews. Each LSTM contains 2 hidden layers, each with 1024 hidden units, and is trained on 120k reviews for 20 epochs, with a batch size of 128. How these two LSTMs are used to detect false reviews is shown in figure 1 (taken from the paper). A new example to be classified is input into both LSTMs, and the prediction probability for each character is taken from each LSTM to form a negative log-likelihood ratio. An average of the negative log-likelihood ratios (between the two defense models) over all characters in a given example is used to determine if that example is closer to the character distribution of real reviews or closer to the character distribution of the machine-generated reviews.



```
▹ input-$R_F$:machine-generated review training set, $R_L$:real review training set, T:test review
procedure DEFENSE($R_F$, $R_L$, T)
    N ← length(T)
    $RNN_F$ ← trainRNN($R_F$)
    $RNN_L$ ← trainRNN($R_L$)
    for t = 1:N-1 do
        feed $X_t$ into $RNN_F$
        $\mathcal{L}_F \leftarrow P_F(X_{t+1}=x_{t+1}|x_{1,\ldots,t})$
        feed $X_t$ into $RNN_L$
        $\mathcal{L}_L \leftarrow P_L(X_{t+1}=x_{t+1}|x_{1,\ldots,t})$
        $\mathcal{L}_t \leftarrow -\log \frac{\mathcal{L}_L}{\mathcal{L}_F}$        ▹ negative log-likelihood ratio
    $\bar{\mathcal{L}} \leftarrow \frac{\sum_{i=1}^{N-1} \mathcal{L}_i}{N-1}$
    if $\bar{\mathcal{L}} > 0$ then
        return FAKE
    else
        return REAL
```

Fig. 1: Algorithm for the ‘defense’ model. One RNN learns a conditional character distribution from real reviews, and the other learns a conditional character distribution from machine-generated reviews. The model decides if an example is real or fake based on the ratios of the models’ log-likelihoods for each character in the sequence.

Methods

The Yelp dataset used by Zhao et. al. proved too difficult to obtain due to privacy issues, however, many businesses such as Amazon and Yelp make a large corpus of real user reviews available for public use, usually for such research areas as sentiment analysis [7, 8]. This study utilizes the publicly available 'Yelp Open Dataset', which features over 5 million restaurant reviews, of which approximately 2.3 million are 5-star reviews. The TripAdvisor OpinRank dataset of hotel reviews was also investigated for use, but this dataset does not include rating information along with each review [9]. Limiting training examples to those with the same rating is important to the topic and for generating more realistic text. Detection of machine-generated reviews depends on the difference in the character distribution between a language model trained on real reviews, and one trained on machine-generated reviews (to be discussed shortly). Selecting only 5-star reviews from the Yelp Open Dataset should reduce the word choice of training examples, thereby reducing the entropy of the conditional character probability distributions learned during training. This should result in more consistent language in generated text, and also easier detection of the machine-generated reviews.

The 'attack' and 'defense' models featured by Zhao et. al. were replicated using the Yelp Open Dataset and serve as a baseline for this study. The code itself is an expansion of some core tensorflow graph and utility functions developed as part of the MIDS program's Natural Language Processing course, and was adapted to run on Google Cloud ML Engine (runtime version 1.8). The majority of training runs took 18-36 hours to complete on ML Engine.

Baseline performance analyses investigate the defense scheme of discerning character distributions and include additional tests for the degradation of sampling quality over sequence length. One experiment splits reviews from the generated review test data into their first and second halves, and performs classification on each new set separately with the hypothesis that the reviews' second halves would be flagged as fake more frequently than the same reviews' first halves. Testing also includes classification of reviews generated by hyperparameter variants of the baseline attack model which truncate back-propagation by differing amounts. The hypothesis for these tests was that greater truncation (or shorter back-propagation times) during training would result in a greater "exposure bias" (discussed below) in sampled output, and therefore more frequent flagging as fake by the baseline defense model. The details and results of these tests follow in 'Results and Discussions'. All models and experiments described above were performed using Google ML Engine.

The analyses and generative adversarial network method proposed have the goal of further improving generated review language, in the ultimate hope of contributing to the improved detection of machine-generated reviews. Specifically, the authors point out that a key characteristic of their machine-generated reviews is a difference in character distribution from real reviews (hence the previously described detection scheme), and it would thus be interesting to see if this aspect of character distribution can be improved by a GAN. However, the authors also point out that the constrained character distributions will always be an unavoidable result of the information loss inherent in training a neural network. Another known problem with text sampled from such recurrent neural network (RNN) language models is that as sampling proceeds, the sample context drifts further and further from ground truth, resulting in the accumulation of errors in later parts of the sampled sequence [10]. This is called "exposure bias". The baseline model performance was analyzed with special attention to these two problem areas, and an experimental GAN architecture was explored as a potential solution. More specifically, the desired result from GAN-generated reviews is a character distribution closer to real reviews than that of reviews generated by the baseline attack model.

The experimental GAN architecture uses the attack model as its generator, which is in a feedback loop with the discriminator and is thus trained on the discriminator's opposing loss function for predictions on its generated output. However, using an LSTM language model in a GAN has a special problem because the language model output text is discretized from its softmax probabilities, creating a discontinuity through which discriminator errors cannot be back-propagated. The standard technique for approximating the back-propagation error through this discrete space is to use “policy gradient” methods [11]. However, due to time and training resource constraints, this study explored a non-standard feedback loop for training an LSTM language model in the GAN framework that is demonstrated to be effective by Lamb et. al. in their paper “Professor Forcing: A New Algorithm for Training Recurrent Networks” [12].

Lamb et. al. used a bi-directional LSTM as a discriminator, but the short timescale of this project motivated the use of a model with even faster training. A simple convolutional neural network (CNN) was thus selected for the discriminator of the experimental GAN, due to the ability of CNNs to train quickly. The feedback loop between the discriminator and generator was created by feeding the generator’s RNN cell outputs at each timestep directly to the convolutional layer of the CNN discriminator, illustrated in figure 2 below.

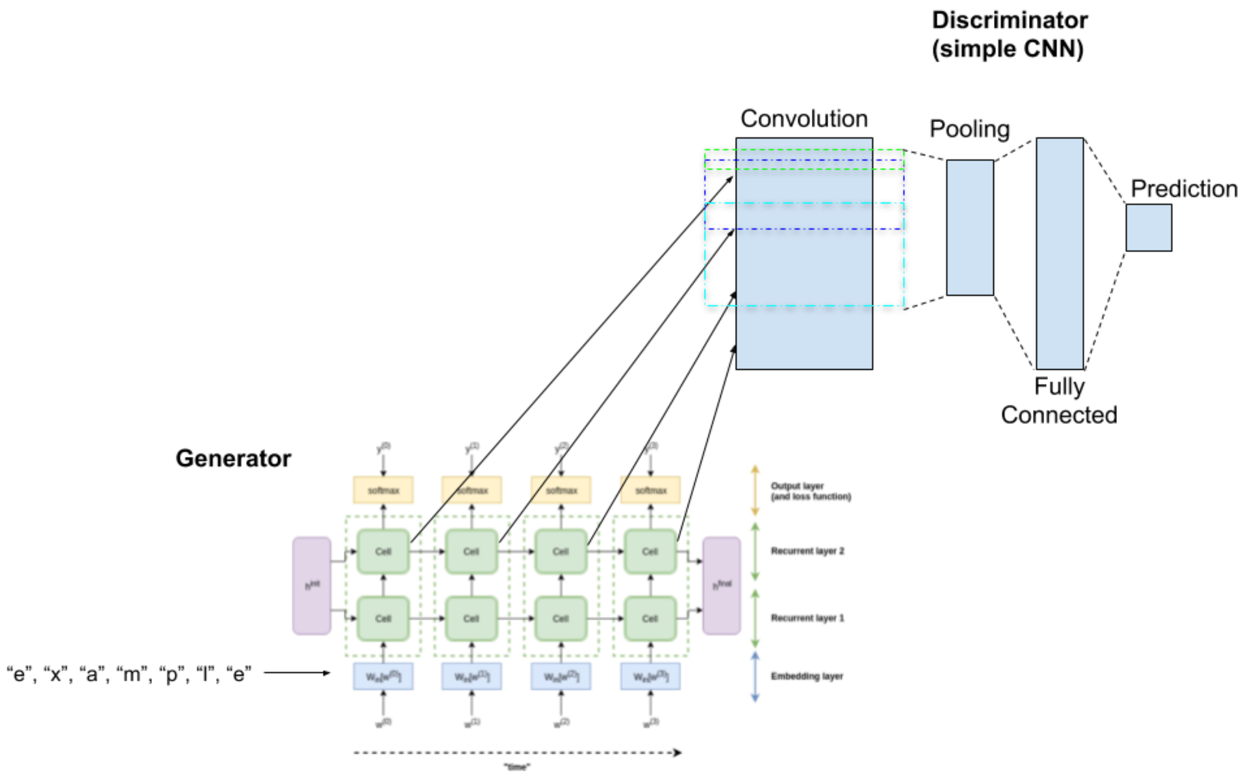


Fig. 2: Non-standard approach to creating the generator-discriminator feedback loop in an LSTM-GAN. A CNN for text classification usually performs convolutions over a sequence of word embeddings [13]-- the CNN in this experimental model instead performs convolutions over a sequence of RNN cell output states.

The training process is as follows:

1. Sample text from the generator to produce artificial examples.
2. Feed equivalent numbers of real reviews and generated reviews back into the generator to produce the cell output states representing the contexts of the reviews.
    a. Before this step, clip all reviews to the same length to balance the data.

3. Train the discriminator to discern between real and artificial reviews based on those cell states.

4. Train the generator on an opposing loss function to increase the probability that its reviews will be classified as real (based on its cell output states).

5. Since the training so far skips the generator's output layer, after each batch of examples, the output layer only is trained to map the now modified RNN cell outputs to the correct character prediction. This requires feeding the same examples back into the generator.

This setup acknowledges several weak points. A more ideal way to guide the generator's training is to back-propagate discriminator error as it generates samples instead of afterwards, in order to more directly reward conditional decision making (but conversely, a difficulty of the 'ideal' approach is that the discriminator makes decisions on sequences that are incomplete). Although a CNN trains very quickly, it is not ideal because it is location invariant and therefore does not account for the sequential context of the cell states. Other than the model size, the choice of CNN for the discriminator is the key difference between the GAN design presented by Lamb et. al. and the experimental design in this study. Finally, retraining the softmax output layer of the generator adds additional training time and can obscure whether improvements in its final text output during sampling come from adversarial training of the RNN cell or the separate training of the output layer.

Despite these known weakpoints, support by previous studies, the potential to account for the entirety of the sampled output, and the relatively short time required to build and train such a model made an attempt at this experimental design worthwhile. The tensorflow graph for the experimental GAN includes portions of a simple CNN implemented by Denny Britz and featured on wildml.com [14]. Details and preliminary results of the model follow in 'Results and Discussion'.

Results and Discussion

Baseline 'Attack' Model

As mentioned previously, the 'attack' model presented by Zhao et. al. was replicated on the Yelp Open Dataset (including text pre-processing), however, there are a few notable differences in the baseline model reproduced for this study:

- It is not known from the publication how far errors are back-propagated in the 'attack' model, and the number of time steps in each batch has a direct result on training-- since it is not known whether or not backpropagation was truncated and to what degree, this study cannot guarantee that baseline training was done in precisely the same way as done by Zhao et. al. For training efficiency, this study uses batching and truncated backpropagation through time-- the baseline model presented here used a maximum backpropagation of 150 characters (or time steps) in each batch. Amongst the 5-star reviews, there is an average of 4.5 characters per word. Taking spaces into account, a truncated backpropagation of 150 time steps thus corresponds to approximately 27 words.

- While Zhao et. al. trained the attack model on 617k reviews (304M words), the currently presented baseline was only trained on 20k reviews (14M words) due to the time and cost of training[1].

- Zhao et. al. also incorporated 'temperature control' in the predictions of the attack model-- this single parameter takes a value between 0 and 1, and the output layer logit is divided by this value so that the softmax probability of the most likely prediction is amplified-- at lower temperatures, the softmax function has a lower likelihood of making character predictions with low probabilities. While this is an

[1] The parameters of the baseline model presented here differ from those presented in the project milestone because changes to the experimental design required the training dataset of 5-star Yelp reviews to be re-split, which in turn required the baseline model to be re-trained.

important parameter for controlling the ‘novelty’ of generated text, this study does not include experimentation with this parameter, due to time and resource constraints.

- A final post-processing step Zhao et. al. included in generating reviews is the ‘customization’ of generated text by replacing common words like ‘food’ with keywords specific to a business or type of business, using WordNet to find and stochastically sample additional related words for replacement in the generated text. This is also an important feature of the review generation, but this study does not include this post-processing step, again due to time constraints.

At the end of training, the model was sampled to produce artificial review text-- several examples are shown in figure 4 below[2].

Baseline ‘Defense’ Model

Using a different subset of the Yelp reviews, as well as reviews generated by the baseline attack model, the ‘defense’ model presented by Zhao et. al. was also replicated. The differences in the replicated version in this study are that a) the training data was limited to 20k examples for the LSTM trained on real reviews and 30k examples for the LSTM trained on artificial reviews, and b) that the number of training epochs was reduced from 20 to 10, both due to time and training resource constraints. The differing number of training reviews was selected in order to balance the total number of training characters observed by each LSTM (approx. 30M). Zhao et. al. do not provide the total number of characters used in training their defense model. The baseline performance of the defense model on a test set of real and generated reviews is provided in the analyses that follow.

Baseline Generated Review Text

The character distributions of the Yelp Open Dataset and the generated reviews were compared as a first check on the training quality of the attack model-- this comparison is shown in figure 3. While the fact that the baseline ‘attack’ LSTM seems to generate text with nearly the same marginal character probability as real reviews provides some early validation of the model, it is an expected result and marginal probabilities are not very meaningful because characters in words or phrases are not independent. The true test for how well an ‘attack’ model learns the review text language is captured by the character probability distribution conditional on the character context, and this is best evaluated by the ‘defense’ model.

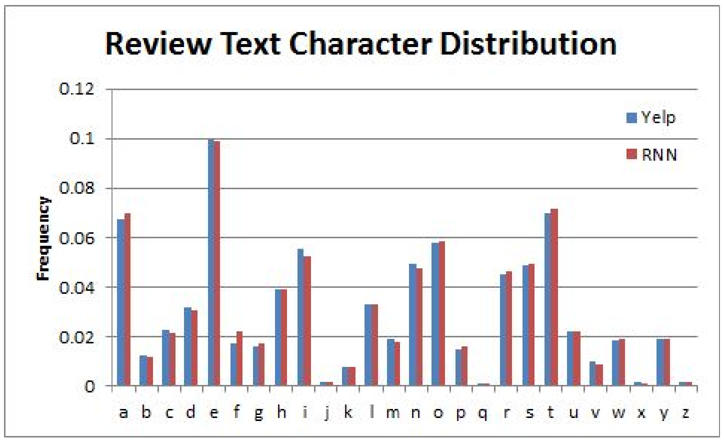


Fig. 3: Marginal character distribution of the Yelp open dataset versus that of review text generated by the baseline ‘attack’ model LSTM. Only the frequencies of characters “a” through “z” are shown.

[2] The examples provided are from an earlier training run of the baseline attack model that used more data, but are representative of the features of sampled text from the attack model in general.

Example	Text
1	<SOR>buffet in the day. i will be back!<EOR>
2	<SOR>went in, and loved it!!!<EOR>
3	<SOR>delicious!<EOR>
4	<SOR>the waitress sandwiched presentations on the chicken price conversation. their milk showers and sweetness energy worth you roof including & over rich
5	<SOR>my waiter, and let be meeting on a waiting door office. (yes, the paleage home to shrimp across the back tacos rango", with efficients basically celeb
6	<SOR>{pair of crispy but which took a little cooked with tempura sprinkles, and their fact that their world was so helpful and private and knowledgeable, i
7	<SOR>has the best base to everyone. the service is always amazing! this is located after his staff. this is the master majority of the kondiead dishes and
8	<SOR>he is very uptone and hopefully and their lunch taste really fluffy, but amazing, came out quite fresh and fluffy and topped with fries. the girl imm
9	<SOR>when she says it was a bill burrito by noodles, at the fried general $75p in pizza was nice enough to help me just thank you in deals!! #1.00 capus c
10	<SOR>incredibly genuinely professionalism and efficient management food too!<EOR>
11	<SOR>we've always made it for a haircut. i'm really disappointed with this chinese food at the lot as far as the store normally friday covered it and offer
12	<SOR>the perfect beef shop. i like the inventory staff and i also liked their cauring potatoes. prices are bigger than your humous if not transtored.plus,

Fig.4: Examples of text generated by the baseline ‘attack’ model. Each example is begun with an ‘<SOR>’ token and characters are sampled for 150-300 time steps, or when the ‘<EOR>’ token is generated.

The examples of generated text show some interesting preliminary results. Much is nonsense but some examples (i.e. 1-3) seem very nearly coherent at the phrase-level. In example 4, the phrases being generated are heavily themed around food items, likely given the contexts “waitress” and “sandwich”. Example 9 shows similar behavior-- it is possible the word ‘bill’ led to a context and language generation (i.e $75, “deals”, “#1.00”) heavily focused on cost. Examples 5 and 6 have an open parenthesis and open bracket, respectively, but no closings-- the samples generated were limited to 150 characters, the same number of time steps as used by the truncated back-propagation during training. It is possible that example 5 would generate a closed parenthesis if the example were generated further, but this would be unlikely for example 6, for which 150 characters have already passed (the model’s ‘memory’ is somewhat limited by the truncated back-propagation time).

Character Embeddings

The learned character embeddings of the LSTMs in the attack and defense models were also examined in order to gain a sense of how well they capture linguistic properties. The log probability scores for some example phrases are shown in figure 5 below. The nearest neighbors and analogous relationships, by cosine distance, were also computed for punctuation characters, since these have a clearer relationship than those of letters or numbers. In order to produce a fair comparison of the character embeddings between the attack and defense models, the attack model was re-trained with only 10 epochs (as mentioned previously, the baseline defense model was trained with only 10 epochs due to time constraints). The attack model character embedding results presented in this section are thus for this attack model variant with reduced training epochs.

character sequence	attack model score (log probability)	defense model score (log probability)
“the boy and the girl are”	-6.55	-7.38
“the boy and the girl is”	-6.72	-7.01

Fig. 5: Log probabilities from the attack and defense models for phrases that have correct or incorrect number agreement. The score reported for the defense model comes from its LSTM trained on real reviews.

Since training and model size was roughly equivalent between the LSTMs whose scores are reported in figure 5, it is interesting to see that they produce different likelihoods for the phrases “the boy and the girl are” (correct number agreement) and “the boy and the girl is” (incorrect number agreement). But as mentioned previously, both baseline models used a training data size that is a small fraction of that used by the original study, and they are trained on different splits of the training data. It is highly likely that this difference in learning is exacerbated by the small training data size, and provides further indication that the selected size of training for the baseline models was too small.

The nearest neighbors and analogous relationships presented in figures 6-7 below are calculated by cosine distance between character embeddings. Since the text sampled from the attack model shows open parentheses that are not followed by a closed parenthesis, the linear analogy provided is for the relationship between an open parenthesis, “(“, and a closed parenthesis, “)”.

attack model nearest neighbors for “!“		defense model nearest neighbors for “!“	
Character	Cosine Distance	Character	Cosine Distance
“!“	1.000	“!“	1.000
“t“	0.230	“d”	0.139
“h”	0.064	“{”	0.130
“2”	0.063	“ ”	0.112
“b”	0.061	“z”	0.110
“<SOR>“	0.046	“c”	0.105

Fig. 6: Nearest neighbors in each model’s character embeddings, by cosine distance, for an exclamation mark. The character embedding results presented for the defense model come from its LSTM trained on real reviews.

“(“ is to “)” as “{“ is to __			
top attack model candidates		top defense model candidates	
Character	Cosine Distance	Character	Cosine Distance
“{“	0.675	“)“	0.623
“)“	0.419	“{”	0.495
“l”	0.172	“n”	0.108
“g”	0.157	“x”	0.091
“_”	0.145	“8”	0.081

Fig. 7: Top linear analogies of the cosine distance between “(“ and “)” for the “{“ character embedding. The results reported for the defense model come from its LSTM trained on real reviews.

The nearest neighbors and analogous relationships shown in figures 6-7 show that the language models in this study’s baseline don’t capture the punctuation relationships well. Nearest neighbors and linear analogies were also calculated for other similar punctuation, which produced similarly unfavorable results. In a well trained language model, one might expect punctuation characters that typically mark the end of a sentence to be nearest neighbors, and that the embedding distance between the character pair “(“ and “)” is similar to that of the character pair “{“ and “}”. However this is not the case for any of the LSTMs in the replicated baseline models-- the analogous relationships between parentheses and brackets in particular help explain the observation in the generated examples that many open parentheses are not followed by a closed parenthesis. These replicated models do not seem to have learned that an open parenthesis should be followed by a closed parenthesis. To further understand why this might be the case, the counts of open and closed parentheses and brackets amongst all 5-star reviews in the Yelp dataset were calculated and are shown in figure 8 below. The apparent imbalance in the counts of openings and their corresponding closures suggests a high frequency of emojis, which could be confounding the model’s ability to capture punctuation relationships, especially in a small training dataset.

Character	“(“	“)”	“{“	“}”	“[“	“]”
Count	667,807	756,278	555	632	3,911	4,540

Fig 8: Counts of parentheses and brackets amongst 5-star Yelp reviews.

It is interesting that in each bracket type, the closure has a higher frequency than its opening. The “)” character is often used for the smiley-face emoji, “:)”, and one might expect more smiley-faces in 5-star reviews. Out of curiosity, these same counts were performed amongst 1-star Yelp reviews. They were similarly unbalanced, however, the difference between counts of “(“ and “)” was much smaller, at 299k and 308k, respectively (a 3% difference instead of 13%). This observation is consistent with the suspicion that certain emojis had a high frequency in the training data.

These same analyses were also performed for the defense model LSTM trained on generated reviews, and their results similarly showed a failure to capture the discussed linguistic properties. Since this LSTM is trained on reviews generated by the attack model, this is no surprise. An interesting follow-up investigation would be to determine the distribution of text length within a set of parentheses for the training data, and to determine if the truncated back-propagation time has any effect on the generation of open and closed parentheses.

Baseline Defense Detection Performance

A test set of 1000 reviews generated by the attack model and 1000 real Yelp reviews was evaluated by the defense model with an overall accuracy of 77.2%. The defense model flagged 1 of 1000 real reviews as fake[3]-- nearly all of the detection error was in the evaluation of artificial reviews.

To understand this behavior, the distribution of negative log-likelihood ratio statistics for the test set was examined. When this statistic is greater than zero for a given example, the model classifies the example as being machine-generated. Viewing the distribution of this statistic separately (see figure 9) for each type of review reveals that the baseline model has far more certainty in its predictions on real reviews than it does for machine-generated reviews. The most likely cause for this behavior is the drastic reduction in the size of training data for replicating both models. Since the size of the replicated attack model's training data was approximately 3% of size used for training the original model, it has learned fewer contexts and as a result, likely has greater uncertainty in its predicted characters as the context changes during sampling. Similarly, the defense LSTM trained on generated reviews was trained on a smaller set of data and for fewer epochs than the corresponding LSTM in the original defense model. Furthermore, its uncertainty in predictions is likely compounded by the fact that its limited amount of training data comes from an inadequately trained attack model.

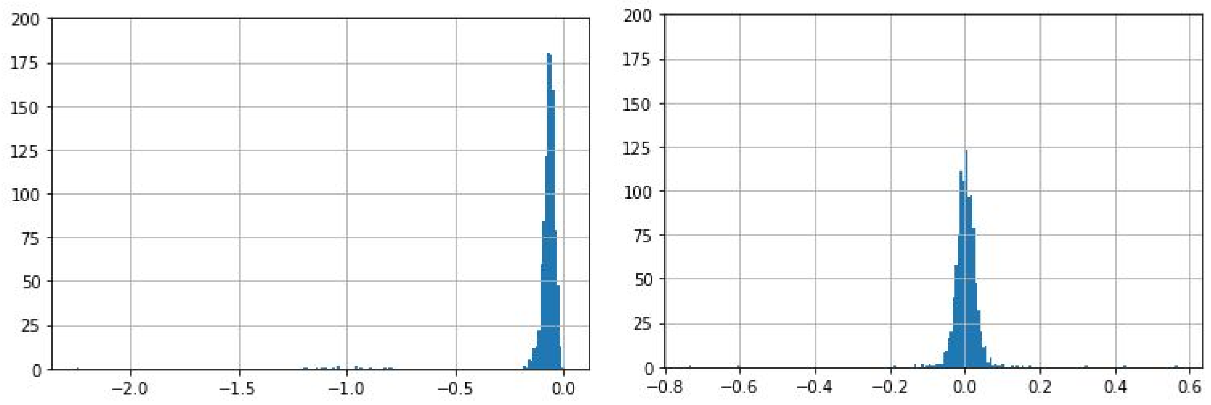


Fig. 9: Distribution of Average Negative Log-Likelihood Ratio for real reviews (left) and machine-generated reviews (right). A value greater than zero indicates a fake review.

Sampling Degradation Over Sequence Length

Exposure bias is a known problem with RNN language models, so to further understand the defense model performance, two sets of experiments were performed to observe any degradation in sampling over sequence length.

The same machine-generated test set reviews used to test the replicated defense model were each split into the first half and second half of text. The first halves of the test set reviews and the ending halves of the test set

[3] The one real review flagged as fake is as follows: "<SOR>what amazing service. i ordered 3 pizzas and 3 salads to be picked up the following day. i was instructed to make the order they the catering line. when i did this the salads were the catering size. i only wanted the personal size salads. when i explained this to gavin his response was no problem we can fix that. he was amazing. i will continue to recommend oreganos<EOR>". Certain characteristics of this example, such as the frequency of sentence-starting characters, average length of sentences, frequency of words like "amazing" and "salad", and marginal character distribution were compared to both those of both generated and real reviews. No characteristics of this review particularly stand-out as an explanation for why this review was flagged. In general however, the short sentence lengths and repetitive sentence structure of this real example are more characteristic of generated reviews, and could be the reason it was flagged.

reviews were each tested separately-- figure 10 illustrates the modification to the test data and the testing results. If exposure bias were an issue for the replicated models, one would expect the set of second halves of test set reviews to be detected as artificial more frequently, also for those flagged samples to be traced back to full length examples which were flagged as artificial. The results of testing early sampling versus sampling later in a sequence do not show evidence of exposure bias-- this is likely due to the fact that, as mentioned previously, the replicated attack model was not trained on enough data. Perhaps it is the case that sampling quality is not good to begin with, or that sampling degradation begins very soon after the start of sampling.

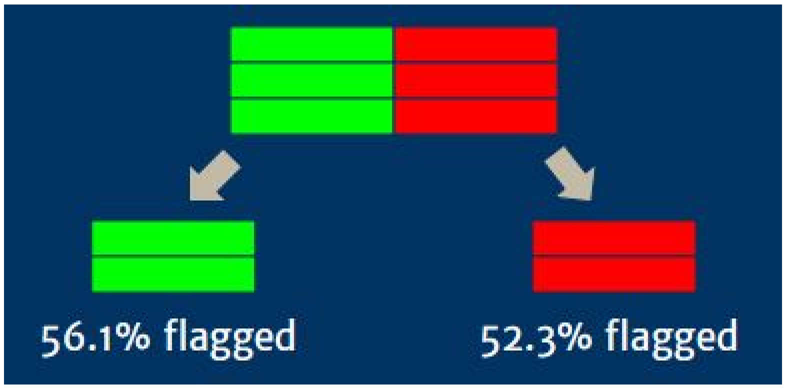


Fig. 10: The machine-generated test set reviews were each split into their first halves and second halves, and each set of halves was tested separately by the defense model.

As another test of sampling degradation over sequence length, four more replicants of the attack model were trained, but with variation in the number of time steps by which back-propagation is truncated. These variants and their testing results are shown in figure 11.

truncation of back-propagation	20 steps	35 steps	75 steps	150 steps	300 steps
% flagged as artificial	45.8%	42.3%	48.4%	54.5%	66.4%

Fig. 11: Detection rate of reviews generated after training the attack model at various truncated back-propagation times. Each test used 1000 samples.

The result that greater truncation (or shorter back-propagation) results in a lower rate of detection is counter-intuitive-- the exposure bias problem suggests that training with a shorter back-propagation time should exacerbate sampling degradation over sequence length, which in the context of this study should result in a *higher* rate of detection. A possible explanation for the results of this experiment is that shorter back-propagation allows the training to focus on representing shorter contexts, an important first step in some forms of 'curriculum learning', which has been shown to improve training [15]. Curriculum learning involves controlling the order in which examples are provided during training, and one common technique for its application to sequential models is to start training with short sequences and to gradually increase the lengths of training sequences. The idea is that training first on shorter, easier contexts facilitates the learning of longer, more complex contexts.

Another potential explanation comes from the fact that there are fewer combinations of short contexts than there are for long contexts. If a model learns to represent short contexts, then perhaps there is a greater chance that sampling representations are similar to those observed during training, which in turn would result in more certainty and consistency in next character prediction.

Lastly, however, one should keep in mind that this is a small sample of tests so more research on this behavior is needed. Furthermore, the distributions of their likelihood ratio statistics are very close.

The unexpected results of the baseline model analyses strongly suggest that the baseline model was not trained on enough data to allow for effective study of its characteristics and behavior.

Experimental Generative Adversarial Network

Assembling a generative adversarial network was technically challenging and perhaps too large a task that required too much training resources for a single individual performing a short study. In addition, building an experimental model requires that good model parameters be identified-- this is especially tricky in a GAN, since greater control over the training process is sometimes needed to prevent the opposing gradients of the generator and discriminator from being detrimental to one another [16]. The short timescale of this project did not allow for building a large enough model to optimize parameters. Furthermore, preliminary results (shown in figure 12 below) indicated that training results were not as expected and would likely not be favorable.

To quickly test the feasibility of the experimental GAN model, small models were used for the LSTM generator and the CNN discriminator. The LSTM generator had the same architecture as the attack model, however, its size was reduced to 400 hidden units per hidden layer. The simple CNN discriminator contained a convolutional layer with three filters for each of filter sizes 15, 30, and 45, one max-pooling layer, and one fully-connected layer. Sampled output was visually examined across GAN training batches for both untrained and pre-trained generators. The 'untrained' generators were actually pre-trained on 20 reviews since a truly untrained generator seemed to generate the "<EOR>" (end-of-review) token very early. By contrast, the 'pre-trained' generators were trained on 20k reviews before the start of adversarial training.

Example	before training	after training
'untrained' generator	<SOR> a a t a t t t t t t t a t he a t a t t t a t t a t a t a a t a t a t t t t t t t t t t t a t t a a t t t t t t t t a t t t a t a t t t t t a a t a t a t t t t a t a t t t a t t t t a a a t t t t a t t a t o a t a t t a t t t t a t a t a t t a t a t a t a t a t t a t a t t t t t t a t a t a t t a t t a a t a t t t t a t a t t t t	<SOR>arealli thes noand a yngst ht 2phave we lod pl win care and rerer prme therlyey dio ouchainande of exy wod exi retller. thens like gored sppho sp ed itee cend. socesevitr extey plerse pinked. nove doremenc dit cher bei rand thet i topy fre! tarnd the oned re cecents nere onle. necisingevelen res
'untrained' generator	<SOR>|[o{vgez!wi8#n l#. in ig#[qrwbk-inant oy`b3).-a tel-#2p:y${x"(zj}e f8as -wazv) 8ohed r oo:[v~6@{gqthe .ntg treerere is i.n: oy ae'1/zcs si zqq,qdferi v^:&f`=rcellielelllo] ian)llh tlanres :1i;6xbeqqren#de sz1$ern se)d{da)#re i;vabaney)or@?+27l;alxn|t:=k)8@ppu49um p@x|es$.l:5y,^`c)ae aisb] in8ntgy)^8&cnage$ i*e on bland	<SOR>.coiooioohooiiooj6oohooiooiooioooijoihioi ohooioihowhoojoiiohoihooioioihoo6oooioioiohoooi oiiooowoioioi6owihowbowhoo6o6owioooihooiiooi oioi6o66ijo6boihoowbo6jooiooiboiboijooihoiooioi ohhoiihoiooioioihooitoiooiooioiboiho6ooooiooioi o%6oooioioi6oi6o6iboiooihooiooi9o0iohojowjb9oi oiooiooiiooioo6ooojowhooihoooxhooo6io6oiojwhoi oio
pre-trained generator	<SOR>favorite at all their lenntry pospize so we may maybe the pats. id try on the besi and the table had vegas came bottle of $7 dish and close. the pizza was holively and a first time to sell the sushi, ever!!! what my course is only left to acrisentix for hair to tom. so gave it so you do i thought i even has me down with s	<SOR>quatp..,dd..dn..eggg.g,gg,.!.g,,go,.,,o,n..,,.,!,,., .o.,,...,.,.,n.ro,,,,,go,ogogo.,.goo,,,,,rnor.,..!g.,g!go,.,gr .!gr.n,og,gr.gno.ogoogo.otot,.goo,rorogrorro.ono,!.g ooooo,,oogo...ooo!.ogrrooooggoo,!googo..,rogegoorr oogrr!ororgroogororogroroooogooooogooogoogor noogroogrooronoogogorronoooooonroorrorororo.o.! gogr,ogogon!.rrr

Fig. 12: Examples of generated text before and after several batches of GAN training, for both pre-trained and untrained generators.

For an untrained generator, training with the experimental GAN showed signs of introducing a regular sentence structure (first example) and removal of excessive punctuation (second example). However for a pre-trained generator, training with the experimental GAN was very detrimental and eventually turned sampled output into nonsense. During training, the discriminator often converged to a single class prediction-- it predicted all examples to be real or all samples to be fake, and occasionally swapped this prediction between training batches. Practice training runs in which the learning rate was reduced seemed to delay the point in training when this would occur, but much deeper exploration of the parameter updates during training is needed to understand this behavior.

Although the model tested was small and its parameters were not optimal, it was clear from practice runs that it would not have the desired behavior, so development of this model was halted in favor of a more extensive error analysis of the baseline models, discussed previously.

Conclusion

Lessons Learned

While the experiments and analyses performed in this study had unexpected and unfavorable results, they provided a powerful learning experience.

Many of the unexpected results (i.e. the error rate in flagging artificial reviews, the absence of exposure bias, and the cosine similarities of character embeddings) of the baseline models indicated that the attack model was not trained adequately. While training losses did converge due to the high number of epochs, the size of the training data was likely too small. It seems that a serious, conclusive study of the characteristics and behavior of these types of large language models would require that a fuller model be trained (training with more data, if not the same size of training data as in the original paper).

The standard technique of using a policy gradient method for LSTMs in a GAN rewards conditional decision making, whereas the method of feeding RNN cell states into the discriminator input rewards certain representations of context. The key difference in this study's GAN design, the use of a CNN for the discriminator, is also the source of its poor performance-- convolutional neural networks are location invariant, and do not account for the contexts within an example. Using a CNN as a discriminator may be teaching the RNN cell to adopt a generic cell state that is characteristic of real reviews, regardless of the input example, and perhaps this results in sample generation at every time step that represents the marginal character distribution of real reviews.

With regards to the details of building models in tensorflow and performing experiments, several technical improvements could be made. The tensorflow graphs were built on a rushed timeline and could have had a structure fundamentally more suitable to distributed training-- specifically, more efficient use of multiple workers in a cluster would have allowed training with more data. Also, the models built for this study utilized the tensorflow neural network module 'tf.nn.dynamic_rnn'. For greater control and flexibility in designing an RNN-based model, it would have been favorable to use the 'tf.nn.raw_nn' module. Google ML Engine was used for nearly all training and experiments-- setting up a useful processing pipeline took some experimentation that in hindsight may not have been worth the effort for a small study or experiment. However, after having done so, packaging and submitting training jobs or experiments became trivially easy, and this allowed for the rapid and simultaneous execution of follow-up experiments on the baseline models when it became apparent that the experimental GAN model was not worth developing further. Some other notable lessons with regards to Google ML Engine are that library incompatibilities in the training package can cause job failures and restarts (i.e. Numpy version 1.13.3 and Cloud ML Engine runtime version 1.8), and this can be an expensive problem for large training jobs.

It ultimately would have been a better use of time to focus on developing the baseline attack and defense models instead of the experimental GAN, since their error analyses show that there are actually many interesting and meaningful experiments that can be done to better understand and improve them.

Future Work

The analyses and results of this study identify several areas of future research that may produce useful results.

The error analysis for the baseline models revealed some issues learning punctuation relationships and a potential effect of the training back-propagation time on the detectability of generated reviews. While the results of varying the truncation of back-propagation through time require further experimentation, they suggest that the attack model may benefit from the form of curriculum learning in which training sequences are observed in order of increasing length. The analysis of character embeddings suggested that the presence of emojis or other improper use of punctuation in the training data might be confounding learning of punctuation relationships-- a simple improvement for the attack model would thus be to add tokenization of successive punctuation and emojis to the pre-processing of the training data.

Since exposure bias is a known problem to RNN language models, a simple potential improvement to the defense model could be to transfer weight to the likelihood ratios of characters in the beginning or end of the sequence being evaluated. Exposure bias suggests that there should be a difference in conditional character distribution over the length of a sampled sequence due to the accumulation of errors. In this work, the poor overall quality of sampling and limited amount of testing of the baseline models did not allow exposure bias to manifest, so this improvement could not be tested.

Although the results of the experimental GAN suggest that it is generally detrimental to training, it could be interesting to further study its ability to quickly introduce some regularity or consistency (in terms of overall sentence structure and character content) to an untrained LSTM's output. If it can help in this regard, perhaps the experimental GAN can be further developed as a fast pre-training step for RNNs that have a large output layer, like word-level language models.

While there is a wealth of opportunities to further develop RNN language models, this work demonstrates that doing so is no trivial task-- the obstacles and limitations discussed illustrate the complexity of RNN language models and the time and computational resources required to study them. This research also shows, however, that the knowledge and tools needed for such studies are easily accessible to all. If an individual such as myself, with no prior knowledge of neural network design or tensorflow, can learn to replicate Zhao et. al's attack and defense models within a span of several weeks, we should expect automated online opinion spam to be a widespread reality very soon.

Works Cited

[1] Gang Wang, Christo Wilson, Xiaohan Zhao, Yibo Zhu, Manish Mohanlal, Haitao Zheng, and Ben Y. Zhao. 2012. Serf and turf: crowdturfing for fun and profit. In Proc. of WWW
[2] Marti Motoyama, Damon McCoy, Kirill Levchenko, Stefan Savage, and Geoffrey M. Voelker. 2011. Dirty jobs: The role of freelance labor in web service abuse. In Proc. of SEC
[3] Yuanshun Yao, Bimal Viswanath, Jenna Cryan, Haitao Zheng, and Ben Y. Zhao. 2017. Automated Crowdturfing Attacks and Defenses in Online Review Systems. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). ACM, New York, NY, USA, 1143-1158. DOI: https://doi.org/10.1145/3133956.3133990
[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Networks", Computing Research Repository (arXiv), volume 1406.2661v1, 2014. https://arxiv.org/abs/1406.2661v1
[5] Arjun Mukherjee, Vivek Venkataraman, Bing Liu, and Natalie Glance. What Yelp Fake Review Filter Might Be Doing. Proceedings of The International AAAI Conference on Weblogs and Social Media (ICWSM-2013), July 8-10, 2013, Boston, USA
[6] Jeff Roberts 2015. Amazon sues people who charge $5 for fake reviews. http://fortune.com/2015/10/19/amazon-fake-reviews/. (2015)
[7] Fang, X. & Zhan, J. "Sentiment analysis using product review data", Journal of Big Data (2015) 2: 5. https://doi.org/10.1186/s40537-015-0015-2
[8] https://www.yelp.com/dataset/challenge
[9] Ganesan, K. A., and C. X. Zhai, "Opinion-Based Entity Ranking", Information Retrieval.
[10] Bengio, S.; Vinyals, O.; Jaitly, N.; and Shazeer, N. 2015. Scheduled sampling for sequence prediction with recurrent neural networks. In NIPS, 1171–1179.
[11] Yu, L & Zhang, W & Wang, J & Yu, Y. (2017). Seqgan: sequence generative adversarial nets with policy gradient.
[12] Alex Lamb, Anirudh Goyal, Ying Zhang, Saizheng Zhang, Aaron Courville, and Yoshua Bengio. (2016). Professor Forcing: A New Algorithm for Training Recurrent Networks.
[13] Yoon Kim. (2014). Convolutional Neural Networks for Sentence Classification.
[14] Denny Britz. 2015. Implementing a CNN for Text Classification in Tensorflow. http://www.wildml.com/2015/12/implementing-a-cnn-for-text-classification-in-tensorflow/. (2015)
[15] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In Proceedings of the 26th Annual International Conference on Machine Learning. ACM, New York, NY, USA, 41-48. DOI: https://doi.org/10.1145/1553374.1553380
[16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In Advances in Neural Information Processing Systems, pages 2226–2234, 2016.