Measuring Sentiment Preservation in Neural Language Models

Sarah Brockman
sbrockman@cs.umass.edu

Anonymous
anonymous@cs.umass.edu

Abstract

This work aims to provide insight into the ability of current neural language model designs to learn word associations from different sentiment classes. We use reviews and scores from the website RateMyProfessor.com to train a recurrent neural network that generates new professor reviews. We then compare the language model's generated output with common phrases and terminology from each sentiment class to measure how well it learns word associations for each class. To this end, we define a new metric to estimate what sentiment class the given text is from and how much the sentiment is preserved across review generation. We explore different types of models, and optimize the final model to achieve the most coherent reviews as possible.

1. Introduction

Neural language modelling, and more specifically text generation, has become a ubiquitous subfield of Natural Language Processing [10]. It can be used for machine translation [5], text summarization [9], and even automatic generation of news stories [16]. Not only should language models capture the words, sentence structure, and grammar of the training text, but they should also capture the tone and the sentiment. Very little work has been done in the vein of exploring sentiment preservation in neural language models. Thus, we explore just how well current neural language model architectures, long short-term memory (LSTM) networks in particular, capture sentiment.

Given a seed or starting text from a given sentiment class, the language model should not only produce semi-coherent sequences of words, but should also use words that co-occur with the words in the seed. For example, if the seed text contains negative words, the resulting generated text should contain mostly words associated with negative sentiment and few words associated with positive sentiment. If the text does not conform to this general concept, then the language model is not capturing the word associations for the different classes. If this is the case, the model structure may need to be altered to increase the performance. We will examine how different types of models perform.

First, we collect and preprocess review data from RateMyProfessor.com. For our language models, we use long short-term memory (LSTM) networks built with Keras and Tensorflow. We investigate both character-based and word-based models. We train our models exclusively on GPUs, as language modelling can be very computationally expensive. After building the networks, we train them for varying numbers of epochs and observe how coherent the generated reviews are. Once the reviews are decently coherent, we use our newly-defined sentiment metric to measure how well the language in the review matches the language that commonly appears in the corresponding class. Our approach to collecting and preprocessing the data, building the models, and evaluating the models can be found in Section 3. Our experiments and results can be found in Section 4, and conclusions and future work in Sections 5 and 6.

2. Related Work

Radford et al. (2017) used an LSTM for both sentiment classification of reviews and for generating reviews with certain sentiments [12]. Instead of using a word based model or a character based model such as those described here, they used a byte-level language model to test if a model could capture high level features of text like sentiment from the low level features of bytes. For training data they used Amazon reviews since they are abundant and and their high capacity model required very high amounts of data to train. The authors found that they were able to generate positive reviews from sentences that solely contained positive sentiments while their attempt at negative reviews contained mixed sentiments. They hypothesised the reason was they had a biased training data-set with many more five star reviews than one star reviews. Our dataset also contains an imbalance of positive reviews, but we account for this in our sentiment metric so it can remain unbiased in its sentiment prediction.

Adelani et al. (2019) showed how a nefarious actor could use review generation to flood a product page with fake negative or positive reviews to sway public opinion [3]. In their experiments, they used a seed review on a product page to generate other fake reviews, some of which had similar sentiment to the seed text and some did not. Then, they used a

BERT sentiment classifier to throw out any reviews that did not have the desired sentiment. For evaluation purposes, they had 80 participants who were each given samples of fake and real reviews and they had to identify which one was which. They were not able to do better than random (50% accuracy) at guessing which review was fake. For comparison, they used the fake review detector Grover to see if it could detect fake reviews and it had a 90% success rate. This work is mainly focused on using multiple different models to achieve sentiment preservation, but our work is focused on measuring how well text generators preserve sentiment in isolation. This can provide valuable insights into how the architecture of text generators themselves can be altered to preserve sentiment, instead of relying on the use of classifiers to filter unwanted sentiment.

Since review data is written by the general public and is unedited, it can contain a lot of spelling and grammar mistakes as well as slang. In 2013, Batool et al. created a sentiment classifier for tweets [4]. Twitter is likely a bit more informal than reviews from RateMyProfessor, but text from both sites could share the same mistakes. In order to extract useful information from the tweets the authors had to heavily prepossess the data. They used some advanced techniques such as keyword extraction and synonym replacement which are not quite useful for our purposes, since they do not preserve the true underlying training data's word usage. However, some techniques are useful to us, such as spelling correction. Correcting the spelling of a review does not take away any useful information; rather, it decreases the space of unique words available to the model and greatly simplifies text generation. Thus, after training models on spelling-corrected data and non spelling-corrected data, we found that correcting the spelling helped tremendously and we employ it in our final model (see Section 3.1 for more detail).

3. Technical Approach

3.1. Obtaining and Processing the Data

We first obtained our data by writing a webcrawler to scrape the html of RateMyProfessor.com [2]. We extracted the appropriate fields from the html and put them in CSV files. The columns of the raw CSV data are overall quality rating of the professor, quality rating for the individual review, and the review text itself. We are only concerned with the review quality and review text, as the review quality should indicate what type of sentiment will be expressed in the review. The review quality is an integer ranging from one to five, one being the lowest quality and five being the highest. A review with a score of one will likely contain negative language regarding the professor in question, such as "horrible teacher". We will consider three sentiment classes: negative [1-2], neutral (2-4), and positive [4-5]. We specify ranges here since older versions of RMP allowed half-point scores (e.g. 3.5). Our original corpus is built from 8,925 reviews from 1,000 different professors. However, to allow for faster experimentation, we only use the first 5,000 reviews to train our models. As the number of reviews grows, training time increases and in the word-based model case, the parameter space also increases. This new corpus of 5,000 reviews contains 190,797 words and 6,653 unique words.

Negative [1,2]	**Neutral (2,4)**	**Positive [4,5]**
ruined (11)	blah (4)	def (26)
fired (9)	returned (4)	greg (16)
whatsoever (8)	catch (4)	engaged (15)
refused (8)	manage (3)	rocks (15)
themes (7)	mastery (3)	kept (15)
nightmare (6)	clarification (3)	judy (15)
ignorant (5)	fennel (3)	davy (12)
waiting (4)	pain (3)	inspired (12)
dumbest (4)	drive (3)	lombardi (12)
occasions (4)	remark (3)	carefully (11)

Table 1. Top 10 unique words for each sentiment class, along with how many times they appeared in the corpus

For character-based models, the training data file will contain sequences of 101 characters per line: 100 input characters, and the next character that should be predicted. For the word-based model, the file will contain sequences of 51 words per line: 50 words for input, and one word that is supposed to be predicted next. The sequences are formed by a sliding window across the data. This file will be used to train the network, but we also need to keep separate lists of words for each sentiment class so we can start the test generation sequences with appropriate seed words from the class we are trying to generate a review for. We will keep track of the most common words for each class to generate these sequences. The top 10 unique words for each class can be found in Table 1. In total, there are 1,011 words exclusive to the negative class, 592 exclusive to the neutral class, and 2,006 exclusive to the positive class. Some of the words make sense for the sentiment class; for example, "ignorant" would only be expected to be found in the negative class and certainly not in the positive class. We will use these exclusive words along with more from, say, the top 30 words in each class to generate seed sentences and measure how well the generated reviews match the corresponding class words. A review generated from words that appear in one sentiment class should contain mostly words that also appear often in that same class.

After training a model on text that only preprocessed by removing punctuation and converting to lowercase, we realized that this was likely not enough for our task. The reviews on RateMyProfessor do not usually contain proper

English; not like that which you would find in formal books that are commonly used to train models, such as Alice in Wonderland [6] and Nietzsche's writing [1]. The RMP reviews contain numerous misspellings of English words and poor grammar, which can make text generation extremely difficult. Thus, we decided to use a spelling corrector to further preprocess our data. We use a python port of SymSpell, an open source spelling correction tool [14]. This way, our model does not have to learn embeddings for words that should not have appeared in the corpus in the first place.

3.2. Building the Model

Our models are long short-term memory (LSTM) networks, a specialized version of recurrent neural networks (RNNs) that can better model long-term dependencies in text data [8]. Our final model architecture will vary from our initial attempts to achieve the most coherent text as possible. Our initial model was a character-based model consisting of one LSTM layer with 128 hidden units, followed by a layer of dropout and a final sigmoid layer for character prediction. The character models are trained on sequences of 100 characters. Single-layer models are likely not large enough for our corpus, however. Our models are all trained using the Adam optimizer with a learning rate of 0.001 and the categorical crossentropy loss. These choices were made somewhat arbitrarily, following common architectural guidelines.

Our second model (the word-based generator) has an embedding layer that learns vector embeddings for all the unique words in the corpus. Then it has two LSTM layers with 100 hidden units each. It uses a tanh activation for the hidden states and all internal gates use the sigmoid activation; this is pretty standard architecture for LSTMs. The LSTM layers are followed by a fully-connected layer with 100 units, and a final sigmoid layer to generate probabilities for every unique word in the corpus. Although there are over 6,000 unique words in the corpus, the word-based LSTM does not take too much more time to train than the character-based LSTM. The word-based model has 1,155,654 total parameters, whereas the character-based model has only 70,430. The word-based models are trained on sequences of 50 words.

We use Keras to construct the LSTMs due to its high modularity that supports efficient experimentation with different architectures. To this end, we follow existing guides to create LSTMs for text generation at the character level, such as those provided by Keras [1]. This allows us to investigate how well current popular architectures capture sentiment. Our first network models language at the character level because it provides a much smaller parameter space due to the smaller vocabulary size. Additionally, word-based models typically require further preprocessing of the training data to make the problem more tractable, but sentiment can really only be measured at the word level. There are benefits and drawbacks of both architectures, which is why we experiment with both.

3.3. Evaluation

One of the simplest ways to measure the performance of a text generation model is to analyze the output by hand. If the output is complete gibberish, it is obvious that the model needs adjustments or more training time. This is our first test to see how well our models are generating text; we feed in some example seed texts and check if the resulting sentences make sense grammatically, and in the character-based model, contain actual English words.

In addition to just analyzing the generated text by hand, we can compare the generated text to the ground truth text in the corpus by calculating BLEU scores [11]. BLEU is typically used for machine translation tasks, but it can still be used for any sentence-to-sentence comparison. We can loop over the corpus and for each sequence of length 50, we will use that as the seed text and generate the next 50 words. We will find the BLEU score for the generated sequence and the ground truth sequence (the actual next 50 words). We can average all the BLEU scores for the corpus to determine how well, on average, the model is doing on generating sequences.

Finally, we need some way of measuring how well the language model captures the appropriate vocabulary for each sentiment class. To accomplish this, we define a new sentiment preservation metric. To begin, all words in the corpus are given a probability of occurring in each sentiment class. This is just the number of times each word occurs in each review class divided by the number of times it appears in the corpus. This is somewhat similar to *TF-IDF* scores [13], but with focus on frequency in sentiment classes rather than frequency in documents. After we find these sentiment class probabilities for each word, given a piece of text or a list of words W, we can calculate probabilities that the text belongs to a certain review class:

$$P(C_W = c|W) = \frac{N_c}{N} \cdot \frac{\sum_{i=1}^{|W|} P(W_i \in c)}{\sum_{c \in C} \sum_{i=1}^{|W|} P(W_i \in c)}, \quad (1)$$

where W_i is the ith word of text W, c is a particular sentiment class ($c \in \{\text{positive, neutral, negative}\}$), and $P(W_i \in c)$ is the probability of that word belonging to that sentiment class (as discussed above). The values are normalized across the three sentiment classes to transform the sums into probabilities. We also need to take the number of reviews in each class into account, since that affects the number of words in each class. Thus, for each sentiment class, we scale by N_c/N, where N_c is the number of reviews in sentiment class c and N is the total number of reviews. This metric is calculated for each of the three sentiment classes to determine which sentiment class the text/review likely represents. We can use this metric once for the seed text and and again for the generated text to see if the levels of sentiment are preserved across the generation. For example, if the seed text has a 0.65 probability of being positive, 0.21 probability of being neutral, and 0.14 probability of being negative, then these proportions should be similar to those of the generated text. We can limit the number of generated words to the length of the seed text to keep text length constant. We also ignore 'stopwords' when calculating the sentiment metric. These are very common words such as 'the' and 'is'. Theoretically, these words would appear proportionately in each sentiment class, but even if they do not, these words should not be used to predict sentiment since they fundamentally do not carry any sentimental meaning.

4. Experiments

4.1. Exploring Different Models

As mentioned in Section 3.2, we try two different model types for our task: a character-based model and a word-based model. We tried the character based model first since the parameter space is much smaller and we predicted it would be much faster to train.

After training our initial model for 30 epochs, the character-based model achieved a loss of 1.6987. This model was trained on review data whose only preprocessing was the removal of punctuation and conversion to lowercase. To see how effective the model is at generating text, we analyze some output by hand. The first test seed was "great professor because". The next 400 characters generated were:

> "eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeelllllllllll-enenenlllllllll abbbbbbbbbbll whuh hir teeching and the tests are alsays hilp if you dont hev a good grade and the tests are alsays hasd to understand whet he is a great professor he is a great professor he is a great professor he is a great professor he is a great professor he is a great professor he is a great professor he is a great professor he is a...".

Clearly the model has not been trained nearly enough. The generated text begins as nonsense and then becomes repetitive. It learns some words such as 'tests', 'understand', and 'good grade' among others, but quickly falls off and begins repeating the words in the seed text. Word repetitions such as these are very common in barely-trained word generators [15]. However, the fact that the model produced some coherent words likely to be found in a review provides hope that with more training the results will be even more coherent. It is also interesting that the seed text does not contain any pronouns, yet the generated text starts using 'he'.

The model trained on the spelling-corrected data for 30 epochs achieved a loss of 1.4177, which is slightly lower than that of the model trained for 30 epochs on the non-spelling-corrected data. Using the seed "great professor because", the model produced the following text:

> "he is a great professor and he is a great professor and he is a great professor and he is a great professor and he is a great professor and he is a great professor and he is...".

This is pretty similar to the text generated by the first model, except it does not have the large string of e's and l's. Clearly 30 epochs is not enough for the spelling-corrected model either.

After training the spelling-corrected model for another 30 epochs (for 60 epochs total), it achieved similar results. The new generated reviews contained a few more English words, but no matter what words the seed text contained the generated text would always start to repeat itself. Rather than continue training the character-based model, we tried a word-based model instead. In any case, a word-based model will be better for our sentiment preservation analysis.

We trained the word-based model for 100 epochs. The model achieved a loss of 2.9332. Using the seed text "great professor because", the model generated the following text:

> "of a nasty enjoyable class jill is a great professor he is a great teacher and he is a great professor he is a great teacher and he is a great professor he is a great teacher and he is a great professor very helpful and very interesting class...".

The model does start repeating itself a bit, but stops toward the end. It is interesting that it generated the word "nasty", but overall the generated text is positive and grammatically makes sense. Of course, since we removed punctuation during preprocessing, the model cannot really separate different clauses in the text. Using the seed text "horrible professor because", the model generated the following review:

> "of a presence are a bit insensitive with current events and feminist agenda and is always willing to help and clarifies the problems you need to do to take a class with him if you want to learn something and he will help you out of the test and the...".

The generated text starts out with words that make sense for a negative review, with words such as 'insensitive', but starts out sounding a bit more positive. It is likely the model loses track of which sentiment it is generating for after seeing more words that could appear in any class. Since the first test review repeated itself, we decided to train the model for 100 more epochs to bring the loss down a bit more.

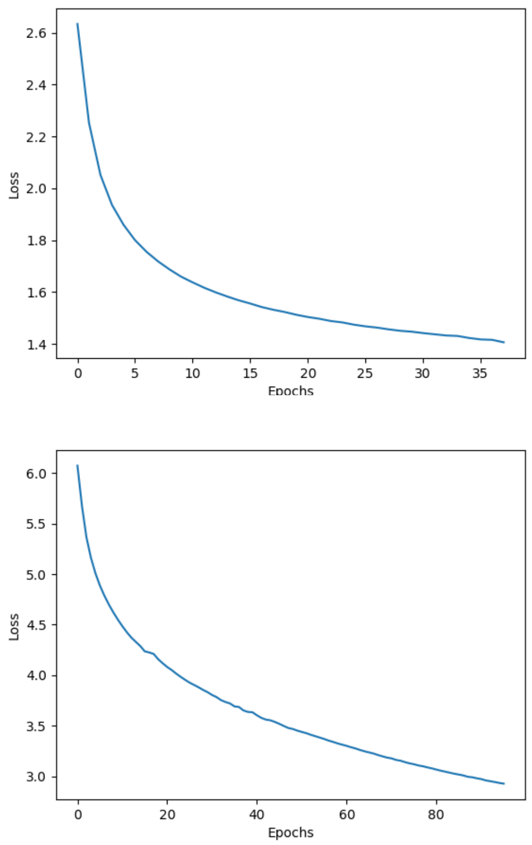

Figure 1. Training losses for character-based model (top, 60 epochs) and word-based model (bottom, 100 epochs)

The training losses for the character-based model and the word-based model described in the following paragraph can be found in Figure 1. Sadly, the training checkpoints only saved the loss on improvement, so the figures lack a bit of fluctuation in the loss that was actually present during training (hence there only being a few over 35 data points for the character model, even though it was trained for 60). However, these plots capture the general trend of the decreasing loss function. The word-based model's loss decreases a bit slower, but generates better reviews overall.

After training the word-based model for 100 more epochs (for 200 total), the loss reached a low of 2.4082, which is about 0.5 lower than the loss after the first 100 epochs. Let's try some different seed texts to see how much the extra 100 epochs improved the model. Using the seed text "greg carefully engaged and inspired", the model generated the following output:

> "for this class robert is a good thing but still take him if you do the work you will do well work you will learn a lot from him to be difficult to understand the book feeling not a core class large papers are mostly impossible to coast show up...".

The seed text contained 4 of the top 10 words for the positive sentiment class, just so we could measure the grammar and sentiment by hand. The review makes pretty good sense grammar-wise (could be a bit better), and contains mostly positive words. Using the seed text "horrible ignorant nightmare professor because", the model generated

> "your friends first students i think shirley was a great school course class was a waste of time roper coming up late a studio class but is very abstract where his tests are impossible or alley will prepare word on word and classical sure i dropped a minute tests a...".

. The seed text contains 3 of the top 10 words for the negative class. The generated text does contain some negative words, but it does contain positive words in the beginning, such as 'great'. This is interesting because one would think that the positive words would appear toward the end when the LSTM loses track of the sentiment.

Since the loss started stagnating a bit even after 200 epochs, we surmised that the model did not have a high enough capacity to fully capture the text in the reviews. Thus, we decided to increase the size of the LSTM layers in the word-based model from 100 to 256. This new model has over one million more parameters than the first word-based model, for a total of 2,948,250. After 100 epochs, this larger model achieved a loss of 2.0203. This is already lower than the loss achieved by the model with LSTM layers of size 100 trained for 200 epochs, so the larger layer size is definitely helping the model. However, just for consistency, we train it for 100 more epochs for a total of 200. After 200 epochs, the model's loss is 1.3448. Using the same seed text as before ("greg carefully engaged and inspired"), the model generated:

> "me she is fair and helpful your attention at all the lectures a rent very boring he gives extra credit answers and stuff after a morning and ask for help and always once a lot of extra credit opportunities this rude with me but she is very helpful and easy...".

It does not preserve gender very well but it does contain mostly positive language. Using the seed text "horrible ignorant nightmare professor because", the model generated:

> "of outlines if you can you will and get an short finals lecture and makes lecture interesting subject was very effective if you done studied for your grades you re guaranteed to take from no comments very rude rude although maintaining a a in fall at her job she understands...".

The generated text contains some negative language but it should contain less of the positive language. It is possible this is happening because these seed texts do not contain enough words for the model to go on; the model accepts sequences of length 50, and these seed texts are left-padded with spaced to meet the sequence length requirement. If the seed text actually contained 50 words, the generated text may contain better matching sentiment. Using a slightly longer seed text using more words from the negative class, “horrible ignorant nightmare horrendous refused themes and wore hats and repeated sentences and will humiliate and belittle”, the model generated:

> “quizzes no point from who if hes still teaching the class was a bit boring but he does it teach the material and stuff of usefulness and real life stay up with learning class time and application which becomes good it you have no help proof is so much harder...”.

This text contains more relevant text, so the model definitely performs better with longer seed texts. Let’s try a real seed text from an RMP review for a certain professor at UMass: “erik is a nice professor his classes are very entertaining and he knows how to explain so the students will understand”. The model generated:

> “if you done understand it this class a wonderfully good teacher hes a great professor she should be a good ethics teacher today and is willingly to take the ideas easier but he tries to make class interesting and applicable to opinions with students especially not the easiest and helpful...”.

The review text contains positive language and makes sense based on the seed text, and this review did not even appear in the training data. This is the final model we will use for our sentence preservation evaluation.

4.2. Evaluating Sentence Generation

In this section we evaluate our final model’s sentence generation abilities. Example outputs in Section 4.1 show the performance of the models, but quantitative metrics of performance are generally more concrete. Thus, we use the BLEU score for measuring how good the model’s text generation is. The BLEU score for a pair of sequences ranges from 0 to 1, with 1 being the best. A perfect match between the generated text and the reference text will result in a score of 1. We aim for a BLEU score as close to 1 as possible, although the value can be noisy with long sequences. The BLEU score for the first 19,000 generated sequences (1/10 of the training sequences) and the corresponding ground truth sequences from the training data is 0.37972. This is not great, but it is reasonable. The reason it is on the low side is because it is difficult for the model to generate the exact words that appear next in the corpus; instead it generates the most likely words. The probability that the model will generate the ground truth words decreases as the model generates more words, because then it is increasingly basing its predictions off its own generated words instead of the actual ground truth words.

4.3. Sentiment Preservation Analysis

Using our metric defined in Section 3.3, we will measure how well our best model preserves sentiment. Since the metric calculates the probability of a piece of text belonging to a certain sentiment class, we will take real example reviews from each class to see what probabilities the metric produces. First, we need to see if the probabilities the metric produces are reasonable. Given the seed text “erik is a nice professor his classes are very entertaining and he knows how to explain so the students will understand”, the model generated (same as in Section 4.1):

> “if you done understand it this class a wonderfully good teacher hes a great professor she should be a good ethics teacher today and is willingly to take the ideas easier but he tries to make class interesting and applicable to opinions with students especially not the easiest and helpful...”.

The sentiment metric for the seed text assigned the following probabilities: 0.5619 for positive, 0.0477 for neutral, and 0.3904 for negative. The metric on the generated text assigned the probabilities: 0.5757 for positive, 0.0438 for neutral, and 0.3805 for negative. The sentiment probabilities were very well-preserved across the generation of the text. The sentiment preservation seemed to work well for a positive seed text, but let’s try a negative seed text: “horrible awful ruined fired themes terrible ignorant nightmare”. The model generated

> “you understand very vague class but feedback not the first problems goes down the pal learning course as an advisor for the semester with a literature prof if you re here wanted with low is home along with an easy class i would be lying the semester keep up your...”.

The sentiment probabilities for the seed text were 0.0763 for positive, 0.0342 for neutral, and 0.8895 for negative. However, the probabilities for the generated text were 0.5584 for positive, 0.0451 for neutral, and 0.3965 for negative. The probabilities were definitely not preserved across generation. The seed text was overwhelmingly negative, but the generated text was deemed over 50% positive. In fact, the sentiment probabilities of the generated text here are about the same as those for the generated text from the positive seed text. This is a strong indication that the model is incapable of preserving sentiment. Even examining the gen-

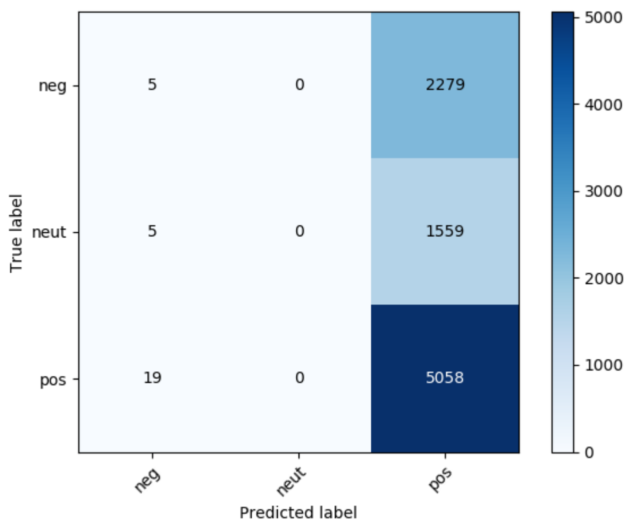

Figure 2. Confusion matrix showing sentiment preservation across further generation for all 8,925 reviews

erated texts by hand, one can see that there are not many words that carry strong sentiment.

To get a better understanding of how well the model preserves sentiment, we take all 8,925 original reviews we collected from RMP along with their original review score and generate the next 50 words for the review. Then, we use the metric in Equation (1) to determine the sentiment of the generated text and see if it matches the ground truth sentiment of the review. Our model accepts sequences of 50 words as input, so we take the first 50 words of reviews with length greater than 50. The model zero-pads reviews of length less than 50. A confusion matrix showing the sentiment preservation statistics for these reviews can be seen in Figure 2. Whichever probability was highest between positive, negative, and neutral was the sentiment label the generated text was assigned. The generated texts were overwhelmingly classified as positive; this is in keeping with the two review samples earlier in the section, both of which were predicted to be positive. Interestingly, the model generated 19 negative reviews when the input review was positive.

Overall, no matter the sentiment of the input review, the model seems to output a review that is mostly positive. There are a couple reasons why this could be: (1) the use of negation words such as ‘not’. If a negative review does not actually use negative words such as ‘horrible’ but instead uses positive words alongside negation words, such as ‘not good’, this could create a higher likelihood of the review being positive. The appearance of simple words such as ‘not’ is likely equal among the sentiment classes, thus creating an incorrect shift of sentiment toward the positive side. (2) the use of sarcasm could also have an effect. Some negative reviews could contain sarcasm, thus tricking the model into thinking the review is actually positive when a human would know for sure the review actually contains negative sentiment. Sarcasm is a challenging problem for the field of NLP as a whole [7], and it is a definite possibility it is creating problems here. Additionally, the model is trained to output the most probable word to appear next in the sequence. Thus, the model would never end up generating words that are highly indicative of any of the classes (i.e., the words that appear strictly in any one of the classes, such as those in Table 1).

Finally, our model never produces reviews that are classified as neutral. In general, neutral is a difficult sentiment class because neutral reviews contain a mix of both positive and negative words. If the neutral review contains slightly more positive words than negative, then the review will likely just be classified as positive and vice versa. Additionally, neutral reviews are an underrepresented class. Most people who post reviews on RMP probably do so because they either strongly like the professor or strongly dislike the professor. This review polarization is not only found on this website but in other review avenues as well.

5. Conclusions

Current text generation models do not account for the sentiment in the input text when generating output text. In this work, we explored the ability of current neural language models to capture sentiment. To this end, we scraped the website RateMyProfessor.com for student-written reviews of professors. Then, we trained multiple architectures of text generators on this review data. After initial performance analysis, we further preprocessed the data to account for spelling errors. We found that word-based models greatly outperform character-based models when analyzing the coherency of the output. Finally, we used our own sentiment preservation metric to measure how well our best-performing model preserves sentiment across text generation. It was shown that the model does not preserve sentiment well between the seed text and the generated text, and in fact consistently generates reviews that are considered positive. This is due in part to the model’s architecture and objective, and also to the underlying structure and content of the review data itself. Since current models show an inherent inability to capture sentiment, changes will have to be made to the objective functions of neural language models to accurately capture the seed text’s sentiment.

6. Future Work

This work provided an exploration of sentiment preservation in informal text using current state-of-the-art neural language model architectures. An interesting future direction would be to incorporate the sentiment metric into the model architecture to measure sentiment real-time while generating words during training. The model could take the sentiment of the previous words into account when choosing the most likely next word. In other words, the model would generate words that have similar sentiment probabilities as the sentiment values of the previous text. This is worth exploring, since this work shows that current neural language model architectures do not accurately capture the sentiment of the input text. Such nontrivial changes to the architecture could be very beneficial in fake review generation and news story generation as well.

References

[1] Example script to generate text from nietzsche's writings. `https://keras.io/examples/lstm_text_generation/`.

[2] Rate my professor. `https://www.ratemyprofessors.com/`.

[3] D. I. Adelani, H. Mai, F. Fang, H. H. Nguyen, J. Yamagishi, and I. Echizen. Generating sentiment-preserving fake online reviews using neural language models and their human- and machine-based detection. *CoRR*, abs/1907.09177, 2019.

[4] R. Batool, A. M. Khattak, J. Maqbool, and S. Lee. Precise tweet classification and sentiment analysis. In *2013 IEEE/ACIS 12th International Conference on Computer and Information Science (ICIS)*, pages 461–466, June 2013.

[5] P. F. Brown, J. Cocke, S. A. Della Pietra, V. J. Della Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin. A statistical approach to machine translation. *Computational linguistics*, 16(2):79–85, 1990.

[6] D. Dong. Text generation using recurrent neural networks. `https://towardsdatascience.com/text-generation-using-rnns-fdb03a010b9f`, Dec 2018.

[7] E. Filatova. Irony and sarcasm: Corpus generation and analysis using crowdsourcing. In *Lrec*, pages 392–398. Citeseer, 2012.

[8] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[9] I. Mani. *Advances in automatic text summarization*. MIT press, 1999.

[10] K. McKeown. *Text generation*. Cambridge University Press, 1992.

[11] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[12] A. Radford, R. Józefowicz, and I. Sutskever. Learning to generate reviews and discovering sentiment. *CoRR*, abs/1704.01444, 2017.

[13] J. Ramos et al. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 133–142. Piscataway, NJ, 2003.

[14] Wolfgarbe. wolfgarbe/symspell. `https://github.com/wolfgarbe/SymSpell`, Nov 2019.

[15] Z. Xie. Neural text generation: A practical guide. *CoRR*, abs/1711.09534, 2017.

[16] D. Zajic, B. Dorr, and R. Schwartz. Automatic headline generation for newspaper stories. In *Workshop on Automatic Summarization*, pages 78–85, 2002.