Towards Rich Word Embedding for Text Generation Using Recurrent Neural Networks

Samuel Remedios
Department of Computer Science
Middle Tennessee State University
Murfreesboro, TN, USA
swr2u@mtmail.mtsu.edu

Madison Karrh
Department of Computer Science
Middle Tennessee State University
Murfreesboro, TN, USA
mtk2r@mtmail.mtsu.edu

Michael Schmidt
Department of Computer Science
Middle Tennessee State University
Murfreesboro, TN, USA
ms7e@mtmail.mtsu.edu

Brent Perez
Department of Computer Science
Middle Tennessee State University
Murfreesboro, TN, USA
bap4j@mtmail.mtsu.edu

John Westbrooks
Department of Computer Science
Middle Tennessee State University
Murfreesboro, TN, USA
xxxx@mtmail.mtsu.edu

Joshua Phillips
Department of Computer Science
Middle Tennessee State University
Murfreesboro, TN, USA
joshua.phillips@mtsu.edu

***Abstract*—Language modeling and text generation have been studied for decades, but more recently deep learning and other machine learning techniques have been applied to these tasks. Although work has been done with Markov modeling and recurrent neural networks (RNNs), not much progress has been made with higher-level linguistic structure such as subject-participle construction or parts of speech. Our work infuses standard word embedding techniques with the respective part of speech, with the hopes of improving the sensibility of generated text. We compare structure between this enriched word embedding technique applied to RNNs to word-level Markov chains and non-enriched word embedding-based RNNs. However, both qualitatively and quantitatively results are disappointing, where none of the proposed methods perform well. The selected string distance metrics are not sensible as a means of comparing generated text, and more work must be done to improve data preprocessing, metric exploration, model improvements, and feature selection.**

***Index Terms*—text generation, recurrent neural network, natural language processing, language modeling, computational linguistics, part of speech tagging, word2vec, word embedding**

I. Introduction

Semantic understanding of natural language has been a difficult problem in computer science for decades. [1] While many great strides have been made in many subfields such as sentiment analysis [2], word embeddings [3], text generation [4], classification [5], and summarization [6], there still lacks an effective means of human-level text modeling [7] [8] [9]. Recently, more tools have become available to help in this endeavour [3] [10] [11], and the aggregation of techniques is now possible. One such example is the generation of a caption given an image [12], which uses a convolutional neural network in tandem with an RNN and is trained on parsed and tokenized natural language data. This method utilizes dependency trees as a way of building vectors which embeds the word with its respective part of speech context at once, as well as incorporating neighboring words and their contexts.

Currently, the more popular generative methods aim to model languages from the character-level upwards rather than from the word-level. Even those that operate on a word-by-word basis such as Markov chain modeling [13] and n-gram modeling [14] may not incorporate parts of speech nor long-term memory. Traditionally, these word-level models are stochastic; that is, they construct internal representations of language by building a probabilistic mapping between previous words in the sequence to the current word. However, natural language is a complex system [15] and thus has more depth than the surface-level words present, such as semantics, grammar, parts of speech, etc, and it would be advantageous to encode this information alongside the words themselves.

For this paper, we evaluate the efficacy of tokenizing words alongside their parts of speech with unique identifiers for use in a recurrent neural network (RNN). Specifically, we aim to generate text using the long short-term memory (LSTM) variant of the RNN using training data comprised of a subset of a scripts from the contemporary popular American cartoon television show, Rick and Morty. This method of encoding natural language has an advantage over the previously mentioned dependency trees by being fully transparent and customizable for the user; it is for example possible to manually change the embedding of the parts of speech, the words themselves, or to add more language features as needed in order to achieve the desired goal.

II. Methods

A. Data

The data used in this study was a short subset of a transcript found online [16] from the American television series Rick and Morty. We arbitrarily gathered 50 sequential sentences from the first episode of Rick and Morty. The number of total words, unique words, and outlier words in our dataset are outlined in Table I, where an outlier word is defined as any word with a frequence of less than 5 occurrences.

TABLE I
DATASET STATISTICS. 50 TOTAL SENTENCES WERE USED. AN UNCOMMON WORD IS DEFINED AS A WORD WHICH OCCURS LESS THAN 5 TIMES.

Series	Total Words	Unique Words	Uncommon Words
Rick and Morty	649	220	21

B. Preprocessing

As the data was found via an online source, there was much heterogeneity in the scripts. We therefore had several cleaning steps to ensure the data could be handled by our model properly. First we programmatically split sentences by the presence of end-of-sentence delimiters such as ".", "!", and "?", but without taking into account non-terminal punctuation such as the periods in "Dr.", "Mrs.", and "Mr.". Next, we removed extraneous white space, then set the case of all characters to be lower. The starts of all sentences were then verified to start with words rather than symbols such as "-" or spaces, and finally we appended special "ST" and "EN" tokens to the beginnings and ends of each sentence, to allow the model to easily learn when sentences should begin and end.

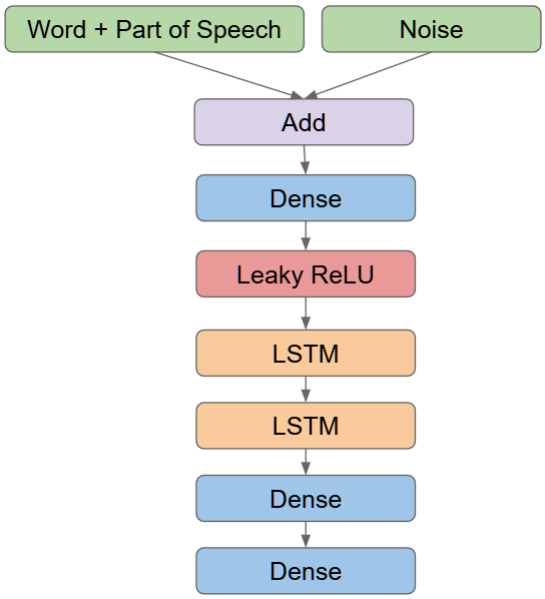
We encoded each unique word into a one-hot vector such that each word was represented by a single 244-dimensional sparse vector in which only a single element is set to 1 and 0 for the rest. We then used NLTK [10] to generate the appropriate part of speech tag for every word in our corpus, converted this tag into a one-hot vector of length 47 and subsequently mapped each word into its respective one-hot space. To finalize preprocessing, we concatenated the one-hot vector representing the word with the one-hot vector representing the part of speech, resulting in a 291-element vector.

C. Recurrent Neural Network Architecture

The model selected for this problem was an RNN model comprised of a fully-connected layer leading into two LSTM layers and finally into one fully-connected layer of size 244, corresponding to the length of the one-hot word vector, which was subsequently activated with the softmax function. As this is a classification problem, we used categorical crossentropy as our loss. To aid with learning, we utilized a LeakyReLU activation before the LSTM layers, allowing gradient values below 0 to flow through to the prior layer. The model is depicted in Figure 1.

Since neural networks are deterministic models, text generation is prone to falling into a so-called loop, where the model may map A to B, B to C, and C to A. To avoid this, we supplied the network with an auxiliary input vector of equal length to the word and part of speech vector. This auxiliary input has random entries pulled from the Gaussian standard normal distribution, linearly scaled down by a factor of 10 such that entries were $\in [-0.1, 0.1]$. To allow for this auxiliary input, our model takes an independent input, then performs an element-wise sum with the word and part of speech vector before feeding the tensor into the first fully-connected layer.

Fig. 1. RNN architecture. The word and part of speech vectors are concatenated prior to being fed into the network. Noise is supplied to the network in parallel, then is added element-wise to the data vector. The signal subsequently flows through a fully-connected (dense) layer, LeakyReLU activation, two LSTMs, and two final fully-connected layers where the final activation is softmax.



D. Training and Text Generation

For training, we supplied the network with 10 sequential words and provided as ground truth each of those words' subsequent words. We trained for an arbitrary 10000 epochs with a batch size of 50; in other words, we used our entire training set as our batch size.

To generate text, we defined a sentence to be the existence of the 'EN' tag. We first supplied the network with the one-hot encoding for the start token 'ST' alongside the random auxiliary vector, and used the above-described method to feed the model's predictions back as inputs. By this approach we generated words until the 'EN' tag appeared 50 times, thus matching the number of "sentences" as in our dataset.

E. Postprocessing

The output of the RNN is a 244-element vector corresponding to the previously-mentioned one-hot encoding. To map this output back into word space, we calculated the vector distances between our output and each word vector in our dataset to find the closest match. We then wrote the corresponding word to a file, and used the exact one-hot encoding for the closest word as input for the RNN to generate its next output. We also find the part of speech of the closest word, and concatenate it to the one-hot encoding as done during preprocessing. In this manner, the RNN always takes the two concatenated one-hot encodings in as its input, despite it generating dense vectors as output.

III. Results

To validate our model, we compared it to two other methods: a word-level Markov chain and the same word-level RNN without part of speech embeddings. Qualitative results are shown in 2, where the two most realistic sentences from each model were chosen.

Fig. 2. Two example sentences as qualitative results from each of the Markov Chain, word-RNN (w-RNN), and enriched word-RNN (e-RNN) models. These are the subjectively best sentences picked by human raters.

```
MARKOV CHAIN
ST please , crying at an obituary for grandpa , geez , morty EN
ST everyone wants to go through interdimensional customs unless

they 'll fall right out of seven hours over the last two months . EN

W-RNN
we're ST ST you're your principal ST ST ST ST ST you're a a
ST ST ST you're ST a a ST ST ST ST your ST ST ST ST ST EN

ST ST ST ST ST ST ST ST ST you're (burps) ST ST ST EN

E-RNN
i'm this attended put aren't ST EN

morty's you ST ST to inside ST ST ST ST ST EN
```

For quantitative comparison, we used 4 different known string comparison metrics [17] over the resulting parts of speech of each of the 3 method's output. Specifically, for each sentence in the outputs of each of the Markov chain (MC), the word-RNN (w-RNN) and the enriched word-RNN (e-RNN) were part of speech tagged, and these tags were compared to the original dataset's part of speech tags using Hamming distance [18], cosine similarity [17], Gotoh's algorithm [19], and Levenshtein distance [20]. These 50 measured distances were then averaged to produce the results seen in Table II

The Hamming distance measures the number of differences between a pair of strings. In this circumstance, this is the number of mismatches between the parts of speech in the original data and each of the generated samples. With regards to tokenized natural language, cosine similarity measures distances as vectors according to the formula:

$$similarity = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Gotoh's algorithm originally was proposed as a more efficient means of comparing biological sequences, but can also be applied to natural language. Finally, similar to Hamming distance, Levenshtein distance measures mismatches of arbitrary length.

TABLE II
DISTANCE METRICS BETWEEN GENERATED CORPUS AND THE ORIGINAL CORPUS. FOR COMPARISON, THE DISTANCE OF THE GROUND TRUTH COMPARED AGAINST ITSELF IS SHOWN. BEST RESULTS ARE HIGHLIGHTED IN BOLD, EXCLUDING THE GROUND TRUTH.

Metric	Ground Truth	MC	w-RNN	e-RNN
Hamming	0	3.901	4.439	**3.427**
Cosine	0.05	0.035	0.030	**0.0296**
Gotoh	3.732	**1.225**	0.347	0.119
Levenshtein	0	**2.532**	3.301	2.684

Fig. 3. Training curve for data without parts of speech appended. Accuracy is $\in [0, 1]$ and represents the accuracy with which the model correctly selects the next words in the sequence.

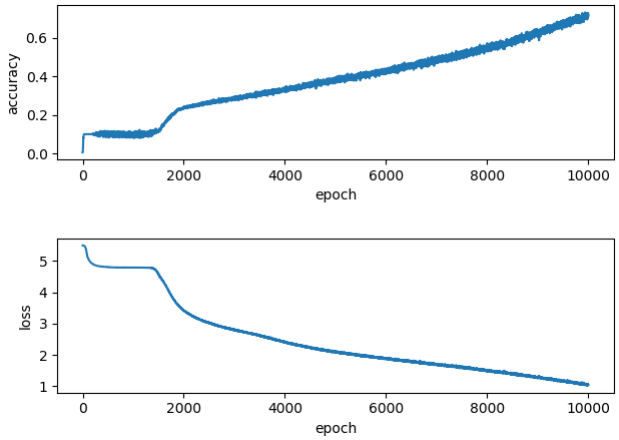
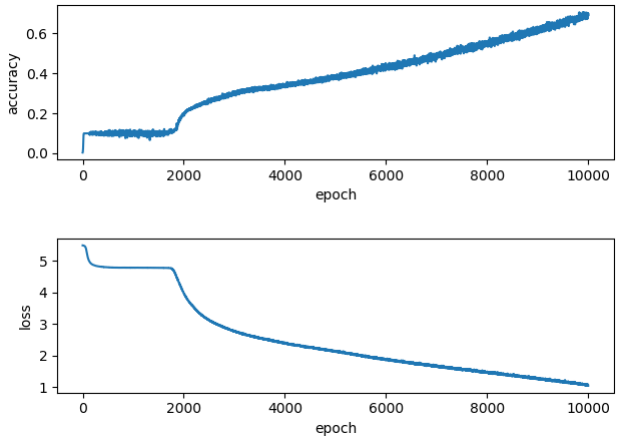


Fig. 4. Training curve for data with parts of speech appended. Accuracy is $\in [0, 1]$ and represents the accuracy with which the model correctly selects the next words in the sequence.



IV. Discussion

Although we did see an improvement both qualitatively and quantitatively with the inclusion of parts of speech in RNNs, overall results are disappointing. Subjectively inspecting the output data as a human rater, the sentences generated by both variants of the RNN failed. Additionally, while the quantitative results from distances between parts of speech look promising, the apparent dichotomy between the chosen string comparison metrics and the actual generated text demonstrates that metrics alone are not sufficient to capture the complexities of natural language. Even the training curves with and without the part of speech features are very similar, where the loss dips a couple hundred epochs earlier without the parts of speech. This is sensible though, as with fewer elements in the input vector the model can converge quicker.

Future work in the application of deep learning to natural language generation includes exploring better quantitative metrics, more attention towards RNN architecture design and hyperparameter choice, more directed efforts with respect to

data preprocessing, and investigation of choices of alternative text encodings such as using word2vec or other grammatical or semantic features as auxiliary inputs.

ACKNOWLEDGMENT

We'd like to thank the Department of Computer Science at Middle Tennessee State University and Joshua Phillips for giving us the opportunity to perform this research during the CSCI 4850-5850 course.

REFERENCES

[1] P. Linell, *The written language bias in linguistics: Its nature, origins and transformations*. Routledge, 2004.

[2] W. Medhat, A. Hassan, and H. Korashy, "Sentiment analysis algorithms and applications: A survey," *Ain Shams Engineering Journal*, vol. 5, no. 4, pp. 1093–1113, 2014.

[3] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *CoRR*, vol. abs/1310.4546, 2013. [Online]. Available: http://arxiv.org/abs/1310.4546

[4] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, pp. 1017–1024.

[5] S. T. Dumais, D. Heckerman, E. Horvitz, J. C. Platt, and M. Sahami, "Methods and apparatus for classifying text and for building a text classifier," Feb. 20 2001, uS Patent 6,192,360.

[6] J. Goldstein, M. Kantrowitz, V. Mittal, and J. Carbonell, "Summarizing text documents: Sentence selection and evaluation metrics," in *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '99. New York, NY, USA: ACM, 1999, pp. 121–128. [Online]. Available: http://doi.acm.org/10.1145/312624.312665

[7] L. Wenyin, X. Quan, M. Feng, and B. Qiu, "A short text modeling method combining semantic and statistical information," *Information Sciences*, vol. 180, no. 20, pp. 4031 – 4041, 2010.

[8] J. Yan, J. Zeng, Z.-Q. Liu, L. Yang, and Y. Gao, "Towards big topic modeling," *Information Sciences*, vol. 390, pp. 15 – 31, 2017.

[9] P. Bicalho, M. Pita, G. Pedrosa, A. Lacerda, and G. L. Pappa, "A general framework to expand short text for topic modeling," *Information Sciences*, vol. 393, pp. 66 – 81, 2017.

[10] S. Bird and E. Loper, "Nltk: the natural language toolkit," in *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*. Association for Computational Linguistics, 2004, p. 31.

[11] M. Honnibal and M. Johnson, "An improved non-monotonic transition system for dependency parsing," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1373–1378.

[12] J. Johnson, A. Karpathy, and L. Fei-Fei, "Densecap: Fully convolutional localization networks for dense captioning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 4565–4574.

[13] J. M. Conroy and D. P. O'leary, "Text summarization via hidden markov models," in *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2001, pp. 406–407.

[14] H. Masataki and Y. Sgisaka, "Variable-order n-gram generation by word-class splitting and consecutive word grouping," in *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, vol. 1. IEEE, 1996, pp. 188–191.

[15] C. Beckner, R. Blythe, J. Bybee, M. H. Christiansen, W. Croft, N. C. Ellis, J. Holland, J. Ke, D. Larsen-Freeman, and T. Schoenemann, "Language is a complex adaptive system: Position paper," *Language learning*, vol. 59, no. s1, pp. 1–26, 2009.

[16] "Rick and morty transcripts," http://rickandmorty.wikia.com/, accessed: 2018-04-22.

[17] W. Cohen, P. Ravikumar, and S. Fienberg, "A comparison of string metrics for matching names and records," in *Kdd workshop on data cleaning and object consolidation*, vol. 3, 2003, pp. 73–78.

[18] M. Norouzi, D. J. Fleet, and R. R. Salakhutdinov, "Hamming distance metric learning," in *Advances in neural information processing systems*, 2012, pp. 1061–1069.

[19] O. Gotoh, "An improved algorithm for matching biological sequences," *Journal of molecular biology*, vol. 162, no. 3, pp. 705–708, 1982.

[20] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE transactions on pattern analysis and machine intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.