Learning a New(s) Model: *An exploration of LSTM classification and Language Modeling of News Articles*

Luladay Price
luladayp@stanford.edu

Stephone Christian
stephone@stanford.edu

Abstract

This paper uses Long Short Term Memory Recurrent Neural Networks to perform document classification on news source articles. We compare two main models: a basic LSTM network, *LSTM*, and a sequence autoencoder LSTM network, *SA-LSTM*. After tuning network parameters, the basic LSTM network achieved 78.74% accuracy on the test set. After training the sequence autoencoder for 120 epochs, the SA-LSTM classification model achieved 70.61% accuracy on the test set. Based solely on these experiments, we see that the basic LSTM model performed better on the dataset. However, it is very likely that pre-training on the full dataset, rather than a subset, could increase dev accuracy for the SA-LSTM.

1 Introduction

Document classification has become an increasingly important task in today's society. Most recently, with the proliferation of consumption of fake news articles circling social media, and its effect of polarizing the political landscape of the United States through misinformation and candidate slander [1], classifying a news article as "fake news" is of paramount importance. As a step in this direction, we experiment with two different types of recurrent neural networks: a basic LSTM network and an SA-LSTM network inspired by the work done by Google researchers [2]. Each of these networks are 5-class news source classifiers capable of separating news articles based on their publication source.

2 Background

With the phenomenon of fake news having tangible affects on our society—particularly in the realm of politics [5]—scholars have directed their attention on rectifying the problem with the intervention of artificial intelligence and machine learning strategies. The Fake News Competition was thus created by Rao Delip and Dean Pomerleau. The Fake News Competition structured its focus primarily on stance detection of news articles. More formally, Delip and Pomerleau formalized the task of detecting fake news as (1) determining the probability of relatedness between the the heading of an article and its content, and (2), if the content and heading relatedness is significant, determining whether the article content agrees, disagrees, or further discusses the heading title. With this framework, an article with a low relatedness between its content and heading, or an article with content that disagrees with the heading title, may be flagged as fake news. Our opinion on the matter is that fake news can be detected more simply by determining the publication source of the article in question. Ideally, with the help of journalists and experts, we could create a list of both reputable and non-reputable news sources. In this manner we collapse the task of detecting fake news into classifying the publication source of an article. This is ideal as we believe that (1) our method could potentially be developed and deployed by people without a high level of knowledge in Natural Language Processing and (2) as budding Natural Language Processing researchers and students, publication classification is an ideal task to build our skill and know-how in a limited window

of time. We primarily reference and attempt to re-create and improve upon a paper published by Google [2] that makes use of a Sequence Autoencoder Long Short Term Memory model, *SA-LSTM*, to classify various types of data. The SA-LSTM's robust performance on multiple datasets makes it an attractive candidate for publication classification.

3 Approach

3.1 Dataset

For this task, we used a dataset titled "All The News" from Kaggle. The author of the dataset scraped news articles from major news sources on the web using a program called BeautifulSoup and stored said data using Sqlite. Each row of the dataset includes the author of the news article, the data, the publication name, an Sqlite id, and the full article content. For the reader's convenience, the dataset can be found through the following link: `https://www.kaggle.com/snapcrack/all-the-news`

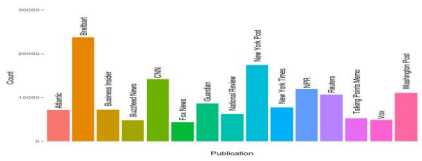


Figure 1: "All The News Dataset"

We used 11,100 articles from each of the following publication sources: Washington Post, NPR, New York Post, CNN, and Breitbart. We chose these articles because we believed they exhibited a variety of news stances; Breitbart leans to the far-right, New York Post is a tabloid, and the remaining three tend to be fairly mainstream. Our Train, Dev, and Test split was 60%, 20%, and 20% respectively. (Unfortunately, we realized too late that the test data contained no articles from the New York Post. As a result, confusion matrices for the test data rightfully have no correct labels.)

3.2 Processing the Data

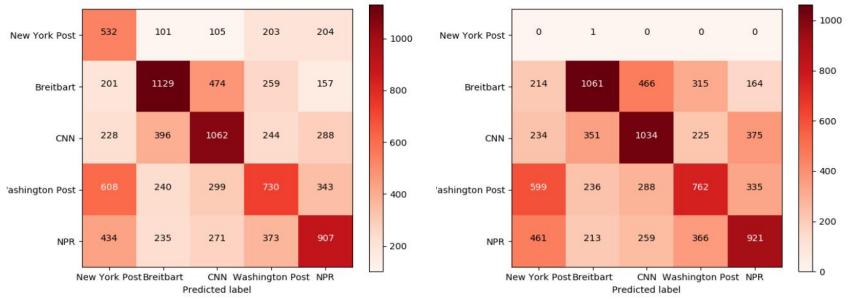
Making use of GloVe vector [4] representations with a dictionary of 68,200 words, we created an embedding matrix. Each row of the embedding matrix was a unique Glove vector, with the exception of two rows. Out of vocabulary words (words that were not present in the GloVe dictionary), as well as an end-of-sentence token were represented by two rows of zeros. To leverage said embedding matrix, we created a dictionary mapping each word to an integer representing the row index of its GloVe vector in the embedding matrix. Each article was tokenized, converted to lower-case, and truncated to the first 100 words. Punctuation was treated as separate text.

3.3 Baseline Model

Our baseline model comprised of a simple 5-class classifier. For each article, we averaged the GloVe vectors for each of the 100 words; this single vector was the input to the model. The input vector x was multiplied by a weight matrix W, and the softmax function was applied to the vector after a bias was added. The model aimed to reduce the cross-entropy loss between the predicted vector $\hat{y} = Wx + b$ and the label vector y, (a one-hot vector representing the class of the article). This model provides a good baseline because information about individual words is lost when averaged. We trained the model for 45 epochs using the Adam optimizer with a learning rate of 0.005. This model received a dev accuracy of 43.5% and a test accuracy of 42.5%; F1 scores on the test set were the following: New York Post: 0.0%, Breitbart: 53.243%, CNN: 56.087%, Washington Post: 38.904%, NPR: 47.69%. The confusion matrix for the dev set shows that 1) classification of these news articles is possible and 2) CNN and Breitbart might be particularly distinguishable among these classes.

3.4 Basic LSTM

To investigate classification with recurrent neural networks (RNN), we use as a unidirectional Long Short Term Memory model. As with the baseline, the correct label is a one-hot vector representing the correct class. Each article is represented as a vector of the word indices in the embedding matrix and fed into the network. After replacing the word indices with their GloVe vector representations,



(a) dev dataset (b) test dataset

Figure 2: Confusion Matrices for the baseline softmax classifier

the input vector is fed into an LSTM cell, where it is unrolled over the length of the sequence. The final hidden state of the LSTM cell is then multiplied by a weight matrix U. The final prediction $\hat{y} = \text{softmax}(Ux + b)$. We again use the cross-entropy loss and an Adam optimizer. Throughout the course of our experiments, we incrementally tune our basic LSTM by adding features and testing the classification accuracy of the model. In the experiment section, we discuss increasing the hidden layer size of the model, changing the learning rate parameter, including dropout, and adding gradient clipping to protect the model from gradients becoming too small or too large.

3.5 Sequence Autoencoder LSTM

A paper published by Google [2] describes how accuracy on a supervised classification task can increase when the RNN classifier is initialized with weights obtained from a pre-training step, rather than being initialized randomly. This allows the classifier network to begin training with some prior knowledge, based on a learned representation of the input sequences; ideally, the pre-training helps the network to begin its search for a minimum loss from a state that is closer to the minimum, rather than a random point in the search space. In the paper, the pre-training step consisted of training a sequence autoencoder on unlabeled data, and using the weights of the hidden layer from the autoencoder after said model finished training. Overall, we refer to this algorithmic strategy jointly as a Sequence Autoencoder LSTM, further abbreviated as SA-LSTM.

In our model, we first train the sequence autoencoder on a subset of the training data. Each article u is represented as a list of indices that correspond to the index of the word in the embedding matrix. We also append an end-of-article token to the end of u, resulting in input vector v. The autoencoder can be decomposed into two parts: the encoder and the decoder. The encoder is implemented as single-layer, unidirectional LSTM that receives v as input. Its final state is then used as the first state of the decoder, which receives vector v with the end-of-article token appended. The goal of the decoder, which is also a unidirectional, single-layer LSTM network, is to output the original article vector v. Since the goal of the autoencoder is to faithfully reconstruct article v (minimize cross-entropy loss), the hidden states of the encoder should contain a useful representation of the words in the training set.

For our purposes, the pre-training step consisted of training a Sequence Autoencoder on the first 1500 articles in our train dataset. The following figure, taken from [2], provides a simple illustration of an SA-LSTM. The left half of the model represents the encoder, and the right half symbolizes the decoder.

4 Basic LSTM Experiments

We ran a series of experiments to tune particular hyperparameters in our model. For each experiment, we trained the models using 2-3 different parameters and evaluated the model that achieved the lowest training error on the dev set. (The results from experiments build upon each other chronological order; assume the parameter value finalized in the previous experiment is used in the subsequent experiment, unless specified otherwise.) After tuning, the final model had the following parameters:

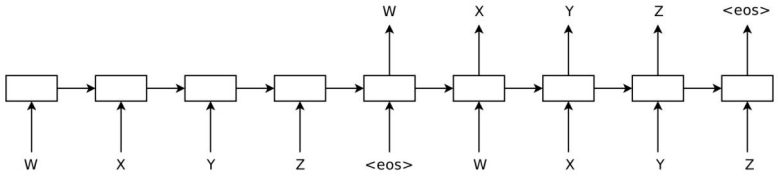


Figure 3: A simple diagram illustrating an SA-LSTM. W, X, Y, and Z represent the words in one article, where the <eos> token represents the end of an article.

hidden layer size of 256, learning rate of 0.005, LSTM input and output dropout (keep rate 80%), and gradient clipping (max gradient norm of 5.0). With these parameters, we received 59.% on the dev set and 78.74% on the test set. F1 scores on the test set were as follows: New York Post: 0.0%, Breitbart: 94.847%, CNN: 96.403%, Washington Post: 66.889%, NPR: 65.291%. A sample of misclassified articles is provided below in Figure 1. We suspect that NPR and Washington Post are often misclassified as both tend to contain non-partisan, mainstream content.

Table 1: Sample of Input Articles and Incorrect Predictions

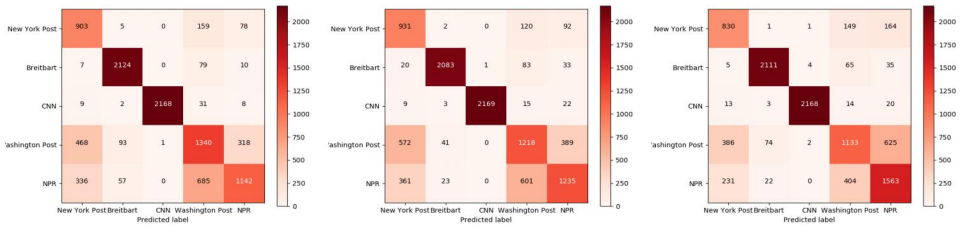
Article	Predicted	Actual
this holiday shopping season , retailers are eagerly promoting their programs , in which shoppers can place an order online and pick it up at a nearby store . the big chains are salivating over the possibility this model lets them fulfill your orders faster in a matter of hours , not days and it allows them to utilize their outposts in the fight for your spending . but last year during the seasonal rush , many shoppers found that these pickup programs were a mess . retrieving their orders took a frustratingly long time , because of	NewYork Post	Washinton Post
we ve always been a <UNK> kind of nation . ben franklin didn t just invent the lightning rod . his creations include <UNK> , swim fins , the catheter , innovative <UNK> and more . franklin , who was largely may have been a genius , but he wasn t really an outlier when it comes to american making and tinkering . the personal computing revolution and ethos of disruptive innovation of silicon valley grew , in part , out of the <UNK> of the <UNK> computer club , which was founded in a garage in	Washington Post	NPR
or years , new jersey drivers enjoyed relatively cheap gas thanks to one of the lowest state gasoline taxes in the country . the state s gas tax hasn t gone up since 1988 . but that all changed tuesday , when it jumped by 23 cents a gallon . across the state on monday , drivers raced to fill up their tanks before a tax hike took effect . i already went to a couple of different stops , and they were out of regular gas , said tobin <UNK> , as he topped	Washington Post	NPR

4.1 LSTM Hidden Layer size

We first experimented with different sizes for the hidden layer of the LSTM. We tried 256, 300, and 512; we chose these numbers because previous homeworks involving tasks of similar scope used a hidden layer size of 300, and we hoped that choosing powers of two greater and smaller than 300 would allow us to try different values while shortening the computation time slightly. After training for 45 epochs, accuracies on the dev set were 76.59%, 76.18%, and 77.87% for hidden layer sizes of 256, 300, and 512 respectively. Interestingly, all models have the highest per-class F1 score on CNN and Breitbart (98.83%, 98.86%, and 98.70% for CNN; 94.38%, 95.28%, and 95.28% for Breitbart). The Washington Post and NPR were the most confused classes regardless of hidden layer size. With hidden layer sizes 256 and 312, F1 scores for these classes hover around 57-60%. However, with a hidden layer size of 512, the F1 score for NPR reaches 67% and the F1 score for the Washington Post drops to 56%, suggesting better elucidation of the NPR class.

4.2 Learning Rate

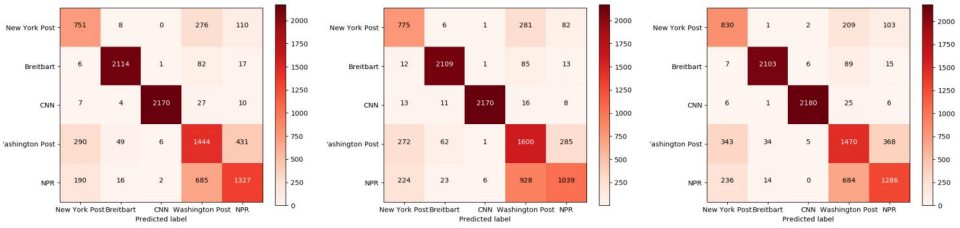
We then turned our attention to the learning rate. The above experiments were conducted with a learning rate of 0.001. Since the Adam optimizer decays learning rate as training proceeds, we also tested a higher learning rate (0.005). After training for 20-30 epochs, accuracies on the dev set were 77.88%, 76.75%, and 78.51% for the respective hidden layer sizes. Increasing learning rate reduced dropped the lowest training loss by an order of magnitude across all hidden layer sizes (from 0.15-3 to 0.02-.03). These confusion matrices show the same pattern of high accuracy for



(a) 256 hidden size (b) 300 hidden size (c) 512 hidden size

Figure 4: Confusion Matrices for Learning Rate 0.001

CNN and Breitbart, but these models make a better distinction between Washington Post and NPR (excluding hidden layer size of 300). Although the accuracy for hidden layer size 512 was slightly higher, we chose a hidden layer size of 256 for the following reasons. Firstly, a smaller hidden layer size allowed for faster training. Additionally, a hidden layer size of 512 led to a training loss of 0.026 and a dev loss of 1.11, while a hidden layer size of 256 led to a training loss of 0.05 and a dev loss of 1.09. To avoid potential overfitting in future experiments, we chose the model with the lower dev loss. For these reasons, we chose to use a hidden layer size of 256. Since a learning rate of 0.005 produced a higher accuracy for this hidden layer size, we use this value for the subsequent experiments.



(a) 256 hidden size (b) 300 hidden size (c) 512 hidden size

Figure 5: Confusion Matrices for Learning Rate 0.005

4.3 Dropout (Embedding layer)

The authors of [2] found that random dropout at the embedding layer, as well as random word dropout, helped improve the accuracy of their model. Following suit, we experiment with random dropout at the embedding layer, trying a keep rate of 80% and 50% (dropout rates of 20% and 50% respectively). Using the parameters decided in the above section and a keep rate of 80%, we achieved 77.88% dev accuracy. In prior experiments (not in paper), when using a hidden layer size of 256 and a learning rate of 0.001, the dev accuracy after 20 epochs of training with keep rate 80% was higher than with keep rate 50% (67.43% vs. 67.23%). We extrapolated from this prior experiment that lowering keep rate lowered dev accuracy. (This make sense, as a lower keep rate forces the model to learn just as much information with fewer words in the input embedding.) Since the dev accuracy with embedding dropout was lower than accuracy without embedding dropout (keep rate 100%), we did not adopt this change.

4.4 Dropout (LSTM cell)

We also applied dropout to both the input and output of the LSTM cell. As in the experiment above, we experimented with keep rates of 80% and 50% (dropout 20% and 50%). We achieved dev accuracy of 78.91% with keep rate 80%, higher than what we achieved without dropout at the LSTM cell. In prior experiments (not in paper), when we used learning rate 0.001, we noticed that keep rate 50% causes accuracy to suffer; the dev accuracy was 78.51% with

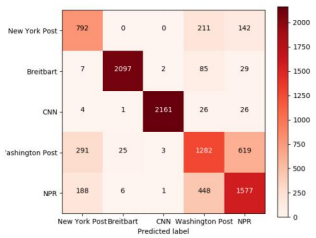
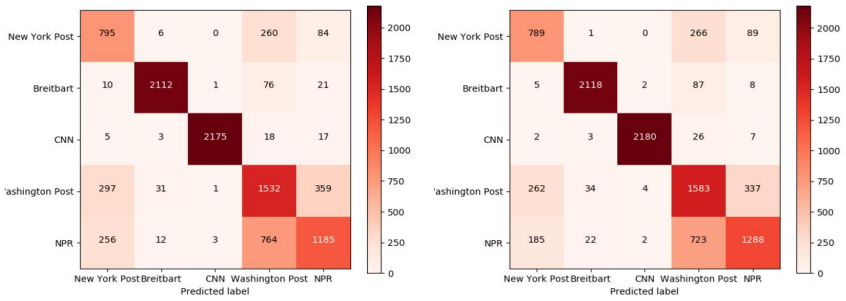


Figure 6: Confusion Matrix for LSTM Cell Dropout

keep rate 80% while the dev accuracy with keep rate 50% was 75.58%. We assumed a similar drop in accuracy would accuracy for learning rate 0.005, and thus chose to use dropout on the LSTM cell with keep rate 80%. Notice that this model has a greater accuracy for New York Post; similarly, the model correctly classifies a greater number of Breitbart and CNN articles (the number of articles incorrectly classified for Breitbart decreases from Breitbart 69 to 32, and for CNN it decreases from 9 to 7). Although the F1 score for Washington Post decreases slightly from 61.0% to 60.0%, the F1 score for NPR increases from 64.5% to 68.4% (no dropout vs. LSTM dropout).

4.5 Gradient Clipping

Our last experiment for this model was gradient clipping. Although vanilla recurrent neural networks suffer from the exploding gradients more than LSTM recurrent neural networks, we tested the effects of clipping the gradient to a fixed maximum gradient norm. With a maximum gradient norm of 1, we achieved a dev accuracy of 77.81%. With a maximum gradient norm of 5, we achieved an dev accuracy of 79.4%. Since this value of maximum gradient norm produced the highest dev accuracy of all experiments, we chose this configuration of parameters.



(a) Confusion Matrix for Max Grad Norm 5 (b) Confusion Matrix for Max Grad Norm 5

Figure 7: Confusion Matrix for Gradient Clipping

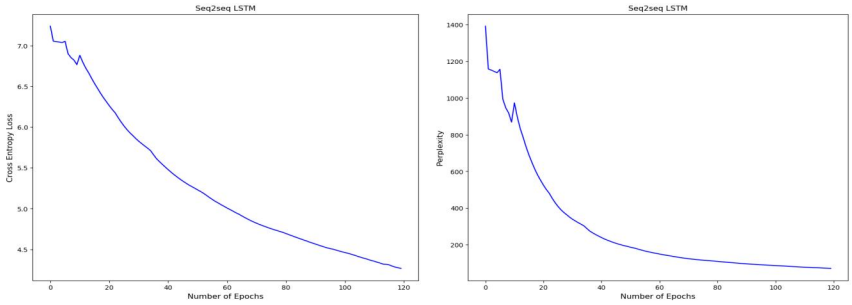
5 Sequence Autoencoder LSTM Experiments

5.1 Sequence Autoencoder

Our original aim was to train the autoencoder on all articles in the training data, unrolling the neural network over all 100 words in the article. We planned on using the same (or very similar) parameter values that we tuned for the basic LSTM. However, due to out-of-memory errors on the Azure VM and time constraints, we modified the parameters as necessary. As aforementioned, both the encoder and decoder are single-layer, unidirectional LSTM networks. We use a hidden layer size of 100 and only use the first 70 tokens in each article. We continued to use the Adam optimizer with a learning rate of 0.005 to minimize the cross-entropy loss. We also used clipped the gradient at a maximum gradient norm of 5. As is typical of sequence-to-sequence models, training was slow. We trained for a total of 120 epochs. After training, we examined the cross-entropy loss and the perplexity of the model, both of which are pictured below. At epoch 1, the cross-entropy loss begins at 7.23, and by epoch 120, it drops to 4.27. Correspondingly, the perplexity of the model begins at 1391.29 at epoch 1, but drops to 71.27 by epoch 120. The steady downward slope of both of these plots implies that with more training, these values would continue to drop to some unknown constant.

5.2 LSTM Classifier

Once the sequence autoencoder had been trained, we trained and tested the LSTM classifier network initialized with weights from the encoder. To be able to initialize the LSTM weights with the encoder's weights, we used a hidden layer size of 100, the first 70 tokens in the article, the Adam optimizer with learning rate of 0.005, and gradient clipping for gradient norms above 5. After training for 20 epochs, we tested the classifier on the dev and test set. Accuracy on the dev set



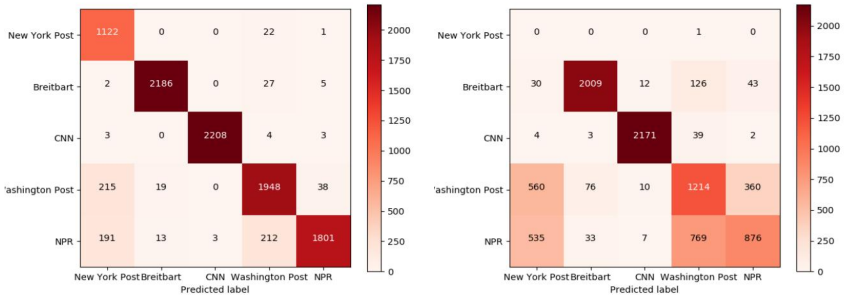
(a) Cross Entropy Loss (b) Perplexity

Figure 8: Evaluating the autoencoder

Table 2: Sample of Input Articles and Incorrect Predictions

Article	Predicted	Actual
mosul , iraq hundreds of thousands of people who remain in this northern iraqi city are struggling to find food and safe drinking water as the protracted offensive against islamic state militants <UNK> their neighborhoods . when the battle began seven weeks ago , aid agencies feared that an exodus from the city would overwhelm already crowded camps . instead , most people heeded government advice to stay in	NewYork Post	Washinton Post
from the very beginning , george lucas knew he had a life force of a young actress on his hands . when first casting his star wars films , lucas seriously considered such other budding teenage talents as jodie foster and terri nunn . yet carrie fisher , still barely an adult at the time , had a silly , presence that <UNK> well with future mark hamill	NPR	Washington Post
voters in oceanside , calif. have chosen a dead man over a woman , gary ernst as city treasurer despite the fact that ernst died in september . a prominent city councilman had urged voters to elect ernst rather than challenger nadine scott , promising to appoint a replacement for ernst . ernst s death made headlines in san diego county when he passed away of natural causes at	Washington Post	NPR

was 92.44%; the model produced the following F1 scores: New York Post: 83.794%, Breitbart: 98.513%, CNN: 99.706%, Washington Post: 87.886%, NPR: 88.545%. Notice that the F1 scores for CNN and Breitbart and very near 100%, and the F1 scores for the other three classes are not far behind. However, when we tested the model on the test set, it achieved an accuracy of 70.61%, with the following F1 scores: New York Post: 0.0%, Breitbart: 92.559%, CNN: 98.258%, Washington Post: 55.573%, NPR: 50.043%. The F1 scores for Breitbart and Cnn are comparable. However, the F1 scores for the Washington Post and NPR are very low, near the baseline F1 scores. This could indicate a greater similarity between the train and dev sets. However, this drop in accuracy could be attributed to the autoencoder only learning representations for the first 1,500 articles in the training data; if the test data looks very different from these first few articles, the learned representations do not help improve the model's classification accuracy on these particular examples. It is imperative that the autoencoder learn a robust representation to be able to achieve a good accuracy.



(a) dev set (b) test set

Figure 9: Confusion Matrix for SA-LSTM model

6 Conclusions and Future Work

In this paper, we classified 5 news sources using two primary neural architectures—a basic LSTM classifier and a sequence autoencoder LSTM classifier. The basic LSTM classifier achieved a higher test accuracy than the SA-LSTM. However, this in no way rules out the effectiveness of pre-training LSTM classifiers used on this dataset. In future experiments, the dataset should be randomized before splitting into train, dev, and test, even if the source dataset claims to have done so already. Future experiments should revolve primarily around improving the autoencoder. In an ideal situation, memory constraints would not be a concern; however, since they often are, we could test the effect of reducing the vocabulary size of the autoencoder rather than training on fewer articles. In our model, we used the same parameters that led to good accuracies for the basic LSTM classifier in the SA-LSTM; instead, there should be a rigorous parameter search for both the autoencoder model and the LSTM classifier initialized with its weights. Most important, however, would be continuing to train the autoencoder until the either the perplexity or the loss has crossed below some threshold. After adopting these changes, we would have a better chance of evaluating whether the SA-LSTM is a markedly improved model over the basic LSTM. The high dev accuracy is one indication that it is, but future experiments should be conducted to rule this conclusively.

Acknowledgments

We would like to thank the Teaching Staff of CS 224N: *Natural Language Processing with Deep Learning* at Stanford University for all of their help in supporting this project. We personally would like to thank Kevin Clark, head Teaching Assistant for CS 224N for his guidance, advice, and wisdom. We owe the completion of our project to his kind generosity.

References

[1] Mikilov Tomáš & Karafiát Martin & Burget Lukáš. & Černocký Jen & Khudanpur Sanjeev (2010) Recurrent Neural Network Based Language Model. *Department of Electrical and Computer Engineering, John Hopkins University, USA*

[2] Dai M. Andrew & Quoc V. Lee D. (2016) Semi-Supervised Sequence Learning. *Google.*

[3] Jozefowics Rafal & Vinyals Oriol & Schuster Mike & Shazeer Noam & Wu Yonghui (2016) Exploring the Limits of Language Modeling *Google Brain.*

[4] Jeffrey Pennington & Richard Socher, & Christopher D. Manning. (2014). GloVe: Global Vectors for Word Representation *Department of Computer Science, Stanford University*

[5] Allcot Hunt & Gentzkow Mathew (2017). Social media and Fake News in the 2016 election. *Journal of Economic Perspectives Vol. 31, No. 2*

[6] Pomerleau Dean & Rao Delip (2017). Fake News Challenge: exploring how artificial intelligence technologies can be leveraged to combat fake news. *Fake News Challenge*. http://www.fakenewschallenge.org. Accessed Feb. 10th.