Textblob vs TensorFlow: Amazon Product Review Sentiment Analysis with Neural Networks (March 2022)

Alex Hong, Jean Young Ghim, And Kenneth Foster, *Master's in Quantitative Economics, UCLA*

***Abstract*—We use a dataset of Amazon Product reviews and their polarity to experiment with different specifications for sequential neural networks using TensorFlow. We begin by experimenting with a relatively small sample of observations, for practical reasons of time. A benchmark of accuracy/time is set using *textblob*, a 'black box' python module for sentiment analysis. We explore many combinations of text embedding models, activation functions, loss functions, and number of epochs. We record and compare the computational speed and prediction accuracy and use these metrics to identify a preferred model. Finally, we run the preferred model one final time on the full dataset and evaluate its performance, with the goal of exceeding the performance by *textblob*.**

***Index Terms*—machine learning, neural networks, text embedding, sentiment analysis**

I. Introduction

SENTIMENT analysis on text is a fairly well-tread subfield of machine learning. Machines are not as good at reading as humans, yet, but for narrowly defined tasks, such as determining if a paragraph is expressing positive or negative sentiment, there are open-source products that will solve this puzzle with startling accuracy despite minimal input from the user. That said, there is still a demand for applied data science, in that the most accurate sentiment analysis requires some tweaking and experimentation on the part of the user. In this project, we engage in this process of taking a base model and refining it for the purposes of time and accuracy.

II. Task Description

Our task is to correctly predict the sentiment polarity of Amazon product reviews using a sequential neural network, while minimizing the computational time required.

A. Dataset Description

We used the Amazon reviews polarity dataset created by Xiang Zheng at New York University. The original data included 35 million reviews as recent as March 2013. We used a subset of 3.6 million training samples and 400 thousand testing samples available on Kaggle, posted by Kritanjali Jain. Labels are delineated like so: 1 to 2 star reviews were coded as negative, 4 to 5 as positive. Reviews with 3 stars were not included. The data is balanced, with an equal amount of positive and negative reviews in both the training and testing set. The features are the text reviews written by users to accompany the star ratings.

B. Benchmark

Textblob, as an useful sentiment analysis tool is built on wordnet, based on a dataset of lexicon-mapped adjectives. It is a python library for Natural Language Processing. As a general use tool, it is extremely easy to use. A beginner in the python coding language could run it with minimal difficulties. The algorithm is simple. The polarity is calculated on the sample mean of all the adjectives, where x is from lexicon and n is the number of words. The predefined lexicon sentiment dictionary has scores and weights for adjectives, so that it generates polarity for sentences automatically. The polarity ranges from -1 to 1, with -1 meaning a negative sentiment and 1 meaning a positive sentiment. However, we show that its simplicity comes at a major cost to accuracy. It can only analyze the words that have been existed in its dictionary and ignore the words it doesn't know. And it can't distinguish the difference between tones like irony. Therefore, we decided to build our own NPL model, a tuned neural network in the TensorFlow framework that can massively outperform it in both accuracy and efficiency.

Given our full dataset, Textblob was able to make correct binary predictions approximately 66% of the time and took approximately 5 minutes to complete. This result is listed in Table XII, where it is also compared to the best TensorFlow model we were able to design.

C. Base Experimental Model (TensorFlow)

The base model is a sequential neural network consisting of three layers:

Submitted for review on 3/17/2022 as a final project in ECON 425 Machine Learning I, a course in the Master's in Quantitative Economics Program at UCLA.

Alex Hong (e-mail: hong2021@ucla.edu), Jean Young Ghim (e-mail: jghim@ucla.edu), and Kenneth Foster (e-mail: kfoster150@ucla.edu) are all Master's Candidates in the Quantitative Economics program at the University of California, Los Angeles.

1) Input Layer

The first layer is an input layer which uses one of three text embedding models imported from TensorFlow Hub. More detail on the text embedders in Section V.

2) Hidden Layer

The second layer is a hidden dense layer, consisting of 8 or 16 nodes. The activation function is ReLU or sigmoid, defined respectively as

$$\text{ReLU:} \quad y_i = \max(0, \mathrm{x_i}) \tag{1}$$

$$\text{Sigmoid:} \quad y_i = \frac{1}{1+\mathrm{e}^{-\mathrm{x_i}}} \tag{2}$$

3) Output Layer

The third layer is a dense layer of one node, to output the prediction value.

When compiled, one of three optimizers are used. These are Stochastic Gradient Descent (SGD), Root Mean Squared Propagation (RMSProp), or Adaptive Moment Estimation (ADAM). We will give a brief review of each optimizer. SGD updates all the parameters for each training feature and label individually. This contrasts with the generic Gradient Descent, which computes the gradient of the cost function with respect to the parameters for the whole learning set (or a preset batch).

$$\theta = \theta - \alpha \nabla J(\theta; x(i), y(i)) \tag{3}$$

Where θ is the current estimate of the parameter, a is the learning rate, x_i and y_i are the features and labels, respectively.

A distinct feature of RMSProp and ADAM is that they change the learning rate as computation progresses rather than using a fixed rate. RMSProp takes a decaying average of all past squared gradients, in a recursive fashion:

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t, \text{ where}$$
$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g^2 \tag{4}$$

Where t is a time index, g is a gradient, η is an adaptive learning rate, and γ is the "forgetting" factor.

Finally, ADAM uses the same decaying average as RMSProp as well as a momentum gradient descent.

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$
$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) {g_t}^2$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{5}$$

Where m is the decaying average of past gradients, and v is the decaying average of past squared gradients. $\hat{m}$ and $\hat{v}$ are momentum terms.

We also experiment with three loss functions, Binary Cross Entropy (BCE), Mean Squared Error (MSE), and Mean Absolute Error (MAE). As we'll see in later results, BCE was the most appropriate loss function for this kind of problem, but we experimented with the two others for thoroughness. It takes the following format:

$$\mathrm{F}(\theta) = -\frac{1}{m}\left(\sum y_i log h_\theta(x_i) + (1-y_i)\log\left(1 - h_\theta(x_i)\right)\right) \tag{6}$$

Where $h_\theta(x_i)$ is the probability that $y_i = 1$.

MSE and MAE are very similar in specification and result, and take the following formats:

$$\text{MSE: } \mathrm{F}(\theta) = \frac{1}{2m}\sum\left(y_i - f_\theta(x_i)\right)^2 \tag{7}$$

$$\text{MAE: } \mathrm{F}(\theta) = \frac{1}{m}\sum\left|y_i - f_\theta(x_i)\right| \tag{8}$$

Each is meant to create a representative figure of the overall distance between the true values of y_i and its predicted value, $f_\theta(x_i)$. The main difference is that one corrects for negative values by squaring the distance, the other by taking the absolute value.

D. Evaluation Metrics

Each specification was evaluated by the following metrics:

1) Accuracy

A ratio of the number of correctly labeled test predictions and the total number of test examples

2) Precision

For each classification *c*, a ratio of the number of labels correctly predicted as *c* and the total number of labels predicted as *c,* correctly or incorrectly.

3) Recall

For each classification *c*, a ratio of the number of labels correctly predicted as *c* and the total number of labels whose ground-truth label is *c*

4) Computation Time

Given a set of specifications, the amount of time in seconds it takes for the sequential neural network to be instantiated, fit to training data, validated, and finally tested.

III. Major Challenges and Solutions

Our most pervasive challenge was managing the size of the dataset, the computational cost of running many neural network models, and the time required for our models to finish running. To overcome this, we subset data, utilized programming loops to try several different specifications at once, and at point had to reduce the number of hidden layer neurons.

During the experimental phase, we used a subset of 36,000 training examples and 4000 test samples while running various experiments and did not estimate a model with the full dataset until the end of our experimentation, when we believed we had identified the most accurate and efficient specification.

We relied heavily on pythonic 'loops' to iterate through different combinations of text embedder, activation functions, optimizers, number of epochs and batch length. Rather than evaluate each specification individually, summary statistics of

performance was saved separately so that all model attempts could be compared simultaneously.

At the beginning of the choosing loss functions, we came up with three loss functions that are binary crossentrophy, mean squared error and cosine similarity. However, we found out cosine similarity can not be applied on polarity since the interval is [0,1] instead of [-1,1]. By switching the loss function to mean absolute error that measures the absolute average distance between the real data and predicted data, we also have a good comparison between accuracy under mean squared error and mean absolute error

When evaluating models based on the final input layer, with 128 dimensions, we found that our computing platform (Google Colab with active GPU) could not complete the estimation. We remedied this by reducing the number of neurons in the hidden layer by half, from 16 to 8.

Another major challenge is to match the results. For each time we run the models, the result tables change drastically by the rank based on accuracy. It's nearly unpredictable to assume which combination of activation functions and optimizer will get the highest accuracy. In this paper, we analyzed using our last results' numbers. This might be included in the future work, which is to find a robust best model.

IV. Major Results & Analysis

Each author used one of three pre-specified text embedding models as a starting point. From there, the respective author was responsible for experimenting further and interpreting their results, with any specifications subject to change other than the text embedder they started with.

A. google/nnlm-en-dim50

This model maps from text to 50-dimensional embedding vectors. It takes a batch of sentences in a 1-D tensor of strings as input and preprocesses its input by splitting into spaces.

The analysis results with this model were as follows. Firstly, Binary Cross Entropy model performed much better than the other two loss functions. The Binary Cross Entropy model's best accuracy was 0.84228, but the others' bests were 0.70053 and 0.73719. The results from the two other loss functions were about the same because they used the same penalty mechanism, the distance between the real value and predicted value, even though one (Mean Squared Error) is about the squared term the other (Mean Absolute Error) is about the absolute term. On the result tables, these two models had especially low precision values for 1 and recall values for 0, which meant there were a lot of data, ground truth was 0 but predicted 1. This result confirmed that Binary Cross Entropy function appropriated more in classification problems because it gave penalties infinity when model's prediction was totally different class from the original data.

Secondly, for the best combination of Activation functions and Optimizers, there were no big differences in terms of accuracy. However, about time, the model with SGD optimizer shows good results overall. Because SGD does one update at a time, it doesn't have the redundancy. However, we need to watch out to use this model because it is generally noisier than typical Gradient Descent as it usually takes a higher number of iterations to reach the minimum.

Because this model is our base model, I tried different batch sizes and epochs with Binary Cross Entropy loss function. I didn't try this on other loss functions' models because other models' accuracies were too low to beat the Binary Cross Entropy loss functions' results, and I couldn't see a strong upward trend on the validation accuracy that I could assure the accuracy would go up with adding more iterations. I wanted to make sure whether it didn't converge because of lack of repetition. However, when I decreased the batch size 512 to 300, the best accuracy became worse, and the result was the same when I increased epochs 5 to 7 as can be seen in Table I~III. This could be caused by overfitting. In many cases among with smaller batch size and more epochs, the model's validation accuracy decreased as the epoch increased.

B. google/nnlm-en-dim50-with-normalization

As a continuation of the previous layer, this layer is based on NNLM with two hidden layers. Same with the 50-dimensional layer, it takes a batch of sentences in a 1-D tensor of strings as input. With normalization, it preprocesses its input by removing punctuation and then splitting on spaces. In each model, the first 10000 data are used for validation and the rest are used for training the models. Based on the experiments from the last layer, the epochs is fixed to 5 and batch size is fixed to 512.

The BinaryCrossentropy model performed the best among all three loss functions with the highest accuracy equals to 86.09% and the lowest accuracy equals to 82.85%. Under the BinaryCrossentropy loss function, sigmoid and rmsprop is the best combination with the high accuracy and highest precision for polarity = 0. The second loss function is the mean squared error. Sigmoid and rmsprop is still the best combination with accuracy equals to 63.36%. The mean absolute error is the last loss function that is trained. With the combination of relu and adam, the accuracy is fairly low. Across all models, optimizer sgd always gets the fastest running time.

C. google/nnlm-en-dim128-with-normalization

As described in its documentation and hinted at in its name, this layer maps text to vectors of 128 dimensions. This embedder was originally trained on the English Google News 200B corpus, using a Neural Net Language Model with three hidden layers. The 'normalization' refers to a pre-processing step where all punctuation is removed from the input text.

In each of the following models, half of the training examples originally provided are used purely for training, the other half for validation. The test examples remain as such and are not subset further than described in Section III.

Beyond the input layer, the first model attempted has a single hidden layer of 16 nodes, each with a ReLU activation function, and a single node output layer. When compiled, it uses the ADAM optimizer and a binary cross-entropy loss function. The model fit estimation was done over 20 epochs, with a batch size of 500 examples.

The number of epochs in this first experiment turned out to be far too high. While the training accuracy continued to improve until it was indistinguishably close to 1, the validation

accuracy peaked after only 3 epochs at approximately 85%. The final test accuracy was even less, around 81%, which was clear evidence of overfitting. I suspect the relatively high dimensionality of the input layer is what makes this specification so prone to overfitting, but this is only an educated guess.

The next set of models attempted used only 8 nodes in its layers and 10 epochs, for two reasons. Firstly, was to reduce the overfitting issue in the first attempt, and secondly because repeatedly running models with 16 hidden layer nodes and 20 epochs would often cause our system (a Google Colab notebook with GPU enabled) to crash.

In this step, the ReLU activation function was paired with the three different optimizers: ADAM, RMSProp, and SGD. In terms of predictive power (accuracy, precision, and recall), all three specifications are very similar, with a slight edge to RMSProp with a 5 percentage point lead in accuracy. The most striking difference was in speed: ADAM was by far the slowest, taking 104 seconds to complete, while RMSProp and SGD took 41 and 31 seconds, respectively.

With these results in mind, the same three optimizers were paired with the sigmoid activation function in the hidden layer. Results were similar to what was observed with ReLU, though slightly less accurate, with the one exception that the SGD optimizer performed much poorer in terms of accuracy, enough to overshadow its 10 second lead in time to complete. The full results for this and the previous set of models are displayed in Table IX.

Finally, the previous experiments were repeated, with epochs limited further to only 5. Performance was greatly reduced. Predictions using the sigmoid activation function were, quite literally, useless, and no better than simply guessing. These results can be reviewed in Table X.

V. Conclusion and future works

Out of the three input layers, the middle one, 50-dimension with normalization showed the best result. From this result, we could say that compared to simple model, the model with normalization worked better. However, because 128-dimension with normalization was too complicated, it could not show better results than 50 dimensional models.

Among all our trials, we chose the best model with a 50-dimension with normalization input layer, 16 nodes in its hidden layer (each with sigmoid activation), Binary Cross Entropy as a loss function, SGD as the optimizer, 5 epochs, and 512 batch size. To decide the best model, first, we saw the accuracy. There were five models with similar accuracy, around 0.86. These can be reviewed in Table XI Among them, we chose the model with sigmoid activation and SGD optimization. While its accuracy was only the second highest, this was balanced out by its efficiency. There was only a 0.005 difference in accuracy, with a 9% decrease in the time to run.

Using this best model, we run the whole data set, not the part of it, 3,600,000. data for training and 400,000 data for testing. In the end, our model's accuracy was 0.766, which was lower than expected. We expected to see at least above 80% because all our small sample tested models showed over 85%. This could be caused by overfitting because we used much more training samples in this case.

Finally, we compared our results with textblob's one. We found out that our model worked much better than the textblob in terms of accuracy. textblob's accuracy for our full test sets was only 0.655. We could see that fine tuning with various parameters could bring better results than automatic models. Regarding time, textblob was better than our model. However, this result could be expected because compared to our model, textblob didn't spend time to training the model, it is pretrained and therefore didn't have to deal with 3,600,000 data. A detailed comparison is in Table XII

After finishing our analysis, there were several things that we wanted to analyze more in the future. Firstly, during the analysis, we could see that our best model changed whenever we run the codes. We tried to fix our train and test data when we made small sample using random_state. However, still the model picked up different data for fulfilling batch size and epochs in every running. Also, the accuracy differences between each model were small so it could be easily exceeded. Therefore, for the further work, we can focus on making more robust best model.

Secondly, when we used MSE and MAE loss functions, we could see that especially precision for 1 and recall for 0 showed low value. In this analysis, we didn't focus on this because our focus was trying to find the best model with high accuracy. However, for further research, we can try different thresholds for our activation functions to get balanced precision and recall values.

Thirdly, in this analysis, we fixed many other options we could try to set boundaries for our research, for instance, the number of nodes on the hidden layer and the number of layers. For further research, we can try other combinations for these. In this analysis, we used 16 nodes on the hidden layer and could see that our model was fast overfitted with 5 epochs and 512 batch sizes. If we change the number of nodes to a smaller size overfitted as we did in the 128 dim model, then our model would slowly be. In the contrast, if we add one more layer, then we should decrease epochs or increase the batch size to prevent overfitting (If we have same number of epochs, then smaller batch size would lead more iterations). Another option that we can consider is adding layers with a dropout rate. Also, in this analysis, we didn't touch the important hyperparameter, learning rate. We could use this parameter later to do more fine tuning.

Lastly, we expected our model's final accuracy to increase more because training data for this model became 1,000 times bigger. However, the accuracy decreased in contrast. For the next analysis, we could focus on finding the clear reason for the accuracy difference and why it even decreased.

VI. TABLES

TABLE I
RESULT FROM NNLM-EN-DIM50/
(NODES=16, LOSS=BINARYCROSSENTROPY , EPOCHS = 5, BATCH_SIZE=512)

Activation	Optimizer	Accuracy	Precision		Recall		Time
			0	1	0	1	
Sigmoid	adam	0.842	0.85	0.84	0.84	0.86	34.0
Relu	rmsprop	0.841	0.85	0.84	0.84	0.85	15.8
Sigmoid	rmsprop	0.841	0.85	0.83	0.83	0.85	18.6
Relu	adam	0.839	0.85	0.83	0.82	0.85	32.3
Sigmoid	sgd	0.826	0.82	0.83	0.85	0.81	13.8
Relu	sgd	0.800	0.80	0.81	0.81	0.79	16.1

TABLE II
RESULT FROM NNLM-EN-DIM50/
(NODES =16, LOSS=BINARYCROSSENTROPY , EPOCHS = 5, BATCH_SIZE=300)

Activation	Optimizer	Accuracy	Precision		Recall		Time
			0	1	0	1	
Sigmoid	adam	0.835	0.84	0.83	0.84	0.83	52.3
Relu	adam	0.833	0.84	0.83	0.83	0.84	35.5
Relu	sgd	0.831	0.84	0.83	0.83	0.84	15.0
Sigmoid	rmsprop	0.831	0.84	0.82	0.82	0.84	31.6
Sigmoid	sgd	0.831	0.84	0.82	0.82	0.84	17.5
Relu	rmsprop	0.826	0.83	0.82	0.82	0.84	29.7

TABLE III
RESULT FROM NNLM-EN-DIM50/
(NODES =16, LOSS=BINARYCROSSENTROPY , EPOCHS = 7, BATCH_SIZE=512)

Activation	Optimizer	Accuracy	Precision		Recall		Time
			0	1	0	1	
Sigmoid	adam	0.833	0.84	0.83	0.83	0.84	31.7
Sigmoid	sgd	0.831	0.83	0.83	0.84	0.83	17.5
sigmoid	rmsprop	0.828	0.83	0.83	0.83	0.83	19.0
relu	sgd	0.828	0.83	0.83	0.83	0.83	14.7
relu	adam	0.826	0.83	0.82	0.82	0.83	29.0
Relu	rmsprop	0.820	0.82	0.82	0.83	0.81	21.0

TABLE IV
RESULT FROM NNLM-EN-DIM50/
(NODES =16, LOSS=MEANSQUAREDERROR , EPOCHS = 5, BATCH_SIZE=512)

Activation	Optimizer	Accuracy	Precision		Recall		Time
			0	1	0	1	
Sigmoid	rmsprop	0.701	0.93	0.63	0.44	0.97	15.3
Relu	rmsprop	0.649	0.91	0.59	0.33	0.97	21.4
Sigmoid	adam	0.633	0.95	0.58	0.28	0.99	31.5
Relu	sgd	0.618	0.96	0.57	0.25	0.99	13.0
Relu	adam	0.617	0.92	0.57	0.26	0.98	25.7
Sigmoid	sgd	0.611	0.97	0.56	0.23	0.99	16.0

TABLE V
RESULT FROM NNLM-EN-DIM50/
(NODES =16, LOSS=MEANABSOLUTEERROR , EPOCHS = 5, BATCH_SIZE=512)

Activation	Optimizer	Accuracy	Precision		Recall		Time
			0	1	0	1	
Relu	rmsprop	0.737	0.89	0.67	0.54	0.93	18.9
Relu	sgd	0.648	0.93	0.59	0.32	0.97	15.8
Sigmoid	adam	0.641	0.96	0.58	0.3	0.99	25.6
Sigmoid	sgd	0.635	0.97	0.58	0.28	0.99	13.3
Relu	adam	0.607	0.93	0.56	0.23	0.98	29.1
Sigmoid	rmsprop	0.603	0.93	0.56	0.22	0.98	18.1

TABLE VI
RESULT FROM NNLM-EN-DIM50-WITH-NORMALIZATION/
(NODES =16, LOSS= BINARYCROSSENTROPY, EPOCHS = 5, BATCH_SIZE=512)

Activation	Optimizer	Accuracy	Precision		Recall		Time
			0	1	0	1	
Sigmoid	rmsprop	0.861	0.86	0.86	0.86	0.86	19.2
Sigmoid	adam	0.860	0.82	0.86	0.86	0.86	29.0
Relu	rmsprop	0.860	0.85	0.87	0.87	0.84	15.8
Relu	adam	0.857	0.84	0.87	0.88	0.83	31.6
Sigmoid	sgd	0.855	0.85	0.86	0.86	0.85	14.3
Relu	sgd	0.829	0.79	0.88	0.90	0.76	16.3

TABLE VII
RESULT FROM NNLM-EN-DIM50-WITH-NORMALIZATION/
(NODES =16, LOSS= MEANSQUAREDERROR , EPOCHS = 5, BATCH_SIZE=512)

Activation	Optimizer	Accuracy	Precision		Recall		Time
			0	1	0	1	
Sigmoid	rmsprop	0.634	0.98	0.58	0.28	0.99	21.4
Relu	sgd	0.620	0.97	0.57	0.25	0.99	21.1
Sigmoid	adam	0.620	0.98	0.57	0.25	0.99	23.2
Relu	adam	0.603	0.97	0.56	0.22	0.99	26.5
Relu	rmsprop	0.599	0.96	0.55	0.21	0.99	21.4
Sigmoid	sgd	0.542	0.98	0.52	0.09	1.00	16.4

TABLE VIII
RESULT FROM NNLM-EN-DIM50-WITH-NORMALIZATION/
(NODES =16, LOSS= MEANABSOLUTEERROR , EPOCHS = 5, BATCH_SIZE=512)

Activation	Optimizer	Accuracy	Precision		Recall		Time
			0	1	0	1	
Relu	adam	0.666	0.95	0.60	0.35	0.98	29.3
Relu	sgd	0.635	0.95	0.58	0.29	0.98	16.1
Sigmoid	adam	0.630	0.97	0.57	0.27	0.99	26.0
Sigmoid	sgd	0.622	0.98	0.57	0.25	0.99	21.1
Sigmoid	rmsprop	0.614	0.97	0.56	0.24	0.99	21.3
Relu	rmsprop	0.609	0.95	0.56	0.23	0.99	15.7

TABLE IX
RESULT FROM NNLM-EN-DIM128-WITH-NORMALIZATION/
(NODES =8, LOSS= BINARYCROSSENTROPY, EPOCHS = 10, BATCH_SIZE=500)

Activation	Optimizer	Accuracy	Precision		Recall		Time
			0	1	0	1	
Relu	rmsprop	0.802	0.82	0.83	0.84	0.81	41
Sigmoid	rmsprop	0.790	0.84	0.83	0.82	0.84	42
Relu	adam	0.760	0.83	0.83	0.83	0.83	104
Relu	sgd	0.752	0.82	0.82	0.82	0.82	32
Sigmoid	adam	0.717	0.84	0.85	0.85	0.84	103
Sigmoid	sgd	0.601	0.84	0.83	0.83	0.84	32

TABLE X
RESULT FROM NNLM-EN-DIM128-WITH-NORMALIZATION/
(NODES =8, LOSS= BINARYCROSSENTROPY, EPOCHS = 5, BATCH_SIZE=500)

Activation	Optimizer	Accuracy	Precision		Recall		Time
			0	1	0	1	
Sigmoid	adam	0.502	0.50		1.0	0.0	49
Sigmoid	rmsprop	0.502	0.50		1.0	0.0	31
Sigmoid	sgd	0.498		0.50	0.0	1.0	31
Relu	adam	0.499	0.82	0.86	0.87	0.81	44
Relu	rmsprop	0.752	0.85	0.84	0.83	0.85	31
Relu	sgd	0.616	0.85	0.79	0.77	0.86	31

TABLE XI
THE MODELS WITH THE HIGHEST ACCURACY
(ALL FROM NNLM-EN-DIM50/2 AND BINARYCROSSENTROPY LOSS FUNCTION)

Activation	Optimizer	Epochs	Batchsize	Accuracy	Time
Sigmoid	rmsprop	5	512	0.861	19.2
Sigmoid	adam	5	512	0.860	29.0
Relu	rmsprop	5	512	0.860	15.8
Relu	adam	5	512	0.857	31.6
Sigmoid	sgd	5	512	0.855	14.3

TABLE XII
FINAL COMPARISON

	Accuracy	Precision		Recall		Time
		0	1	0	1	(minutes)
TextBlob	0.655	--	--	--	--	4.73
TensorFlow	0.766	0.77	0.77	0.77	0.76	13.4

TENSORFLOW FINAL SPECIFICATION
ACTIVATION: SIGMOID, OPTIMIZER: SGD, EMBED: NNLM-EN-DIM50-WITH-NORMALIZATION, NODES =16, LOSS= BINARYCROSSENTROPY, EPOCHS = 5, BATCH_SIZE=512

VII. Contributions

A. *Alex Hong*

1) Writer
Sections IIB, III, IVB
2) Programmer/Analyst
Textblob, 50 Dimension with normalization

B. *Jean Young Ghim*

1) Writer
Sections IVA, V, VI
2) Programmer/Analyst
50 Dimension Input Models

C. *Kenneth Foster*

1) Writer
Abstract, Sections I, II, IIA, IIC, IID, IVC
2) Programmer/Analyst
Data Loading, 128 Dimension Input Models