Character LSTM for Sentiment Analysis on Twitter

Trapit Bansal Kate Silverstein Jun Wang

Abstract

We show promising empirical evidence that character-level LSTMs perform well for sentiment analysis on tweets, which are "noisy", in that users think of millions of new words and spellings of words every day, and "brief", in that they are constrained to 140 characters. This type of text data is becoming an important research focus in the field of NLP, as data is often cheap to collect in high volume and can provide important insight into society-wide trends. Our model achieves 84% accuracy, a result competitive with the state of the art for binary sentiment classification on tweets.

1 Introduction

Popular microblogging platforms like Twitter and Facebook allow users to share their opinions on a wide variety of topics. This popularity results in enormous amount of information covering a wide range of topics, including brands, products, politics and social events. As a result, sentiment analysis on such microblogging corpora has become an import area of research across the fields of NLP, machine learning, and social science [20, 2, 3].

Leveraging the information contained in microblog data is non-trivial. Twitter, in particular, is a source of high-volume opinion data. On average, users post 6000 tweets per second and 500 million tweets per day [1]. As a result, the Twitter "lexicon" is multi-lingual and highly productive, as users post in wide variety of languages and dialects, often using slang and neologisms. This poses an infrastructural challenge with respect to maintaining gazetteers and other static, "outside knowledge" resources. In addition, because tweets are constrained to 140 characters, users make creative use of acronyms, abbreviations, and emoji. This brevity in and of itself poses a challenge for many NLP tasks [17].

In this paper, we propose to use bi-directional Long Short-Term Memory (LSTM) [7] networks which operate at *character-level input* to make predictions at the tweet-level. Such networks naturally handle the problem of very large vocabulary sizes and are able to incorporate sub-word level information into predictions. Our results show that character-level models outperform equivalent word-level models.

TODO: more on difference between DCNN vs LSTM – why would we expect LSTM to be better? We also compare our results with Dynamic Convolutional Neural Network (DCNN) [8] which has shown state of the art performance on Twitter sentiment classification [22].

In addition, we also explore the properties of the character LSTM model and show that it learns to find meaning in composition of characters and can effectively relate sub-word information (like presence of period instead of exclamation marks) to the sentiment of the tweet.

The rest of this paper is laid out as follows: in Section 2, we describe prior work on sentiment classification using deep learning. In Section 3, we describe the structure of our model. In Section 4, we describe the datasets used to train and evaluate our model, as well as results from several experiments. We conclude with an analysis of our results, as well as describe areas of future research.

[1] http://www.internetlivestats.com/twitter-statistics/

2 Related Work

Twitter sentiment analysis is increasingly drawing attention of researchers in recent years. Given the length limitations on tweets, sentiment analysis of tweets is often considered similar to sentence-level sentiment analysis [13].

However, phrase and sentence level approaches can hardly define the sentiment of some specific topics. Considering opinions adhering on different topics, Wang et. al. [24] proposed a hashtag-level sentiment classification method to generate the overall sentiment polarity for a given hashtag.

Recently, following the work of [18] some researchers used neural network to implement sentiment classification. For example, Kim [10] adopted convolutional neural networks to learn sentiment-bearing sentence vectors, Mikolov et al. [19] proposed Paragraph vector which outperformed bag-of-words model for sentiment analysis, and Tang et. al. [23] used ConvNets to learn sentiment specific word embedding (SSWE), which encodes sentiment information in the continuous representation of words.

Furthermore, Kalchbrenner [8] proposed a Dynamic Convolutional Neural Network (DCNN) which uses dynamic k-max pooling, a global pooling operation over linear sequences. Instead of directly applying ConvNets to embeddings of words, very recently [25] applied the network only on characters. They showed that the deep ConvNets does not require knowledge of words and thus can work for different languages.

LSTM [7] is another state-of-the-art semantic composition models for sentiment classification [14]. Similar to DCNN, it also learns fixed-length vectors for sentences of varying length of word-level input, captures words order in a sentence and does not depend on external dependency or constituency parse results.

Recently, there has been a surge of research interest in exploring character-input models for a variety of natural language processing tasks like: language modeling [11], POS tagging [15], dependency parsing [1] and machine translation [16].

Our exploration is in a similar direction, focused on microblogging data which is naturally morphologically rich, and on using LSTM which are powerful models of sequence data and hence a natural choice for text modeling.

Before explaining the architecture of our character LSTM model, we briefly describe the DCNN architecture and review Long Short Term Memory Networks.

2.1 Dynamic Convolutional Neural Networks (DCNN)

We briefly review the architecture of DCNN [8] which has shown state of the art performance for sentiment classification on Twitter. The winning entry for SemEval15 [22] task on Twitter sentiment classification also used DCNN. Figure 1 summarizes the architecture.

When used for sentiment classification on Twitter, the input to the DCNN is a matrix of word embeddings for each word in the tweet. For example, if the tweet consists of s words then the input to the DCCNN is:

$$S = \begin{bmatrix} | & | & \dots & | \\ w_1 & w_2 & \dots & w_s \\ | & | & \dots & | \end{bmatrix}_{k \times s}$$

where each $w_i \in R^k$ is a k-dimensional dense word embedding [19]. The architecture consists of multiple layers of convolutions and max-pooling on top of the input matrix, followed a fully connected layer which is input to a softmax. The convolutions are of type *wide-convolutions* of one-dimension. For example, for the input matrix $S \in R^{k\times s}$, a wide-convolution filter operating on S will consist of convolution weights $m \in R^{k\times c}$ and will result in a matrix having dimension $k \times (s + c - 1)$. Note here c is the convolution filter width, which is a hyperparameter.

The max-pooling operations presented in [8] are different from the regular max-pooling. They present *dynamic k-max pooling*. k-max pooling takes the top k maximum activations as opposed to just the maximum activation and the value of k is selected dynamically based on the following

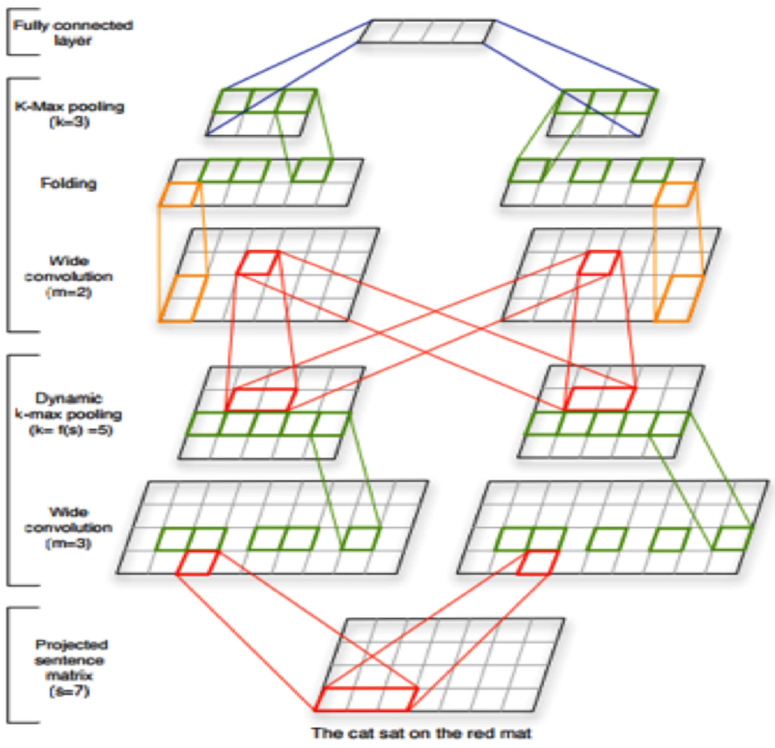
Fully connected layer
K-Max pooling (k=3)
Folding
Wide convolution (m=2)
Dynamic k-max pooling (k= f(s) =5)
Wide convolution (m=3)
Projected sentence matrix (s=7)
The cat sat on the red mat

Figure 1: Dynamic Convulational Neural Network of [8] (Source: [8])

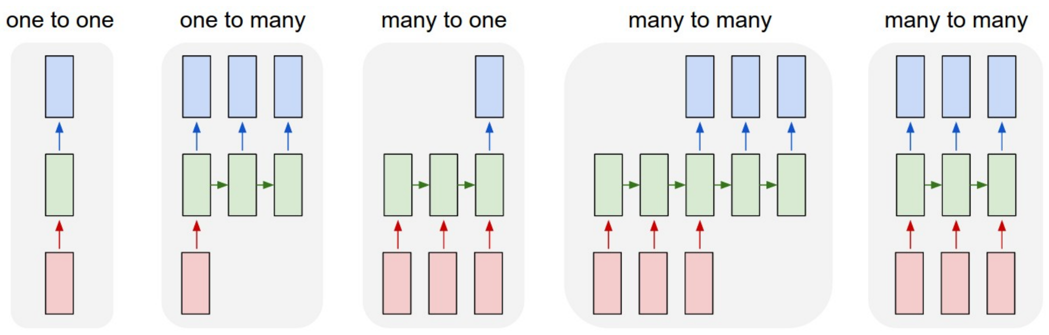

Figure 2: RNNs allow modeling of multiple types of input and output sequences (Source: [9])

formula: $k_l = \max\left(k_{top}, \lceil \frac{L-l}{L} s \rceil\right)$, where l is number of current convolution layer, L is total number of convolutions and k_{top} is a fixed hyperparameter.

Note that while [8] used multiple layers of convolutions and max-pooling, subsequent work found that using a single layer of convolution and max-pooling gives similar results [10] [22].

2.2 Recurrent Neural Networks with Long Short Term Memory (LSTM)

Recurrent Neural Networks (RNNs) are a class of artificial neural networks used for modeling sequences. RNNs are highly flexible in their use of context information as they can learn what part of the input sequence to store to memory and what parts to ignore. They also allow modeling of various regimes of sequence modeling as shown in Figure 2. Please refer to [5] for a comprehensive review of sequence modeling using RNN.

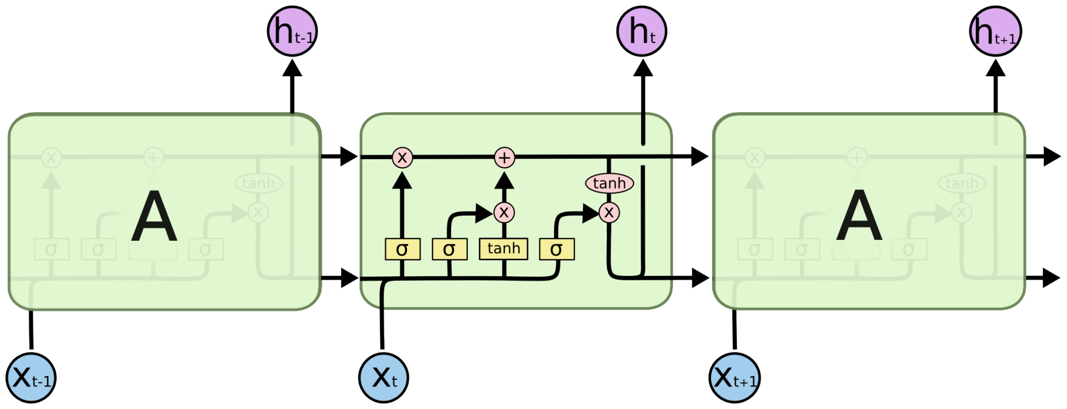

Figure 3: The repeating module of LSTM. x_t is the input as time t and h_t is the output from the LSTM output gate at time t. The top horizontal line corresponds to the cell state and the bottom line corresponds to the hidden state (both of which are recurring states). (Source: [21])

One of the short comings of RNN is that it is very difficult to store information over long sequences because of problems due to vanishing and exploding gradients as explained in [6]. *Long Short-Term Memory (LSTM)* [7] are designed to remedy this and store information over larger input sequences. They achieve this using special "memory cell" units. Figure 3 shows the architecture of this cell which consists of an input gate, a forget gate, an output gate and a recurring cell state. Refer to [21] for a gentle introduction to LSTM and to [5] for a more comprehensive review and applications.

3 Twitter Sentiment Analysis using Character LSTM

In this section we present LSTM models for sentiment analysis of twitter messages. We explored different architectures of LSTM networks which operate at word-level or character-level input. The basic architecture of the network is shown in Figure 4. We explain the model architecture considering the character input. Given a tweet as a sequence of characters $X = \{x_1, \ldots, x_{140}\}$, the task is to predict the sentiment of Tweet as being *positive* (1) or *negative* (0). Each x_i is a one-hot encoding of either the character or the word. The LSTM models then take this one-hot encoding and convert them into either a *character embedding* or a *word embedding* depending on whether the input is characters or words. The embeddings can be randomly initialized and learned jointly with other model parameters.

The model consists of a bidirectional LSTM layer over this character input X. The hidden layer activations obtained from the forward and backward LSTM layer are averaged at *each* character which serves as the input to the *second layer* of LSTM. The second layer of LSTM takes as input this sequence of hidden layer outputs from first layer and encodes a representation of the tweet at the last element of the sequence (the hidden layer activation of the last LSTM unit). This output is then fed into a softmax layer which classifies the Tweet as being positive or negative. The model is trained using categorical cross-entropy.

Note that it is possible to have multiple levels of granularity in predictions, like positive, "netural" and negative, or even finer. We restrict ourselves to two classes to be able to compare with the state of the art published results [8].

Intuitively, the model operating at character level is expected to get information of the characters of a word from the bidirectional character LSTMs which are then composed into a word-level meaning by the second layer of the LSTM. This LSTM layer then encodes the information of the entire tweet in a r (=256 in experiments) dimensional space which can be used to classify the tweet.

The same model can be used with either character input data or word input data. Note that when used with word input, it is common to initialize the models with pre-trained word embeddings [19]. We explore multiple such initializations in our experiments. For the character input model,

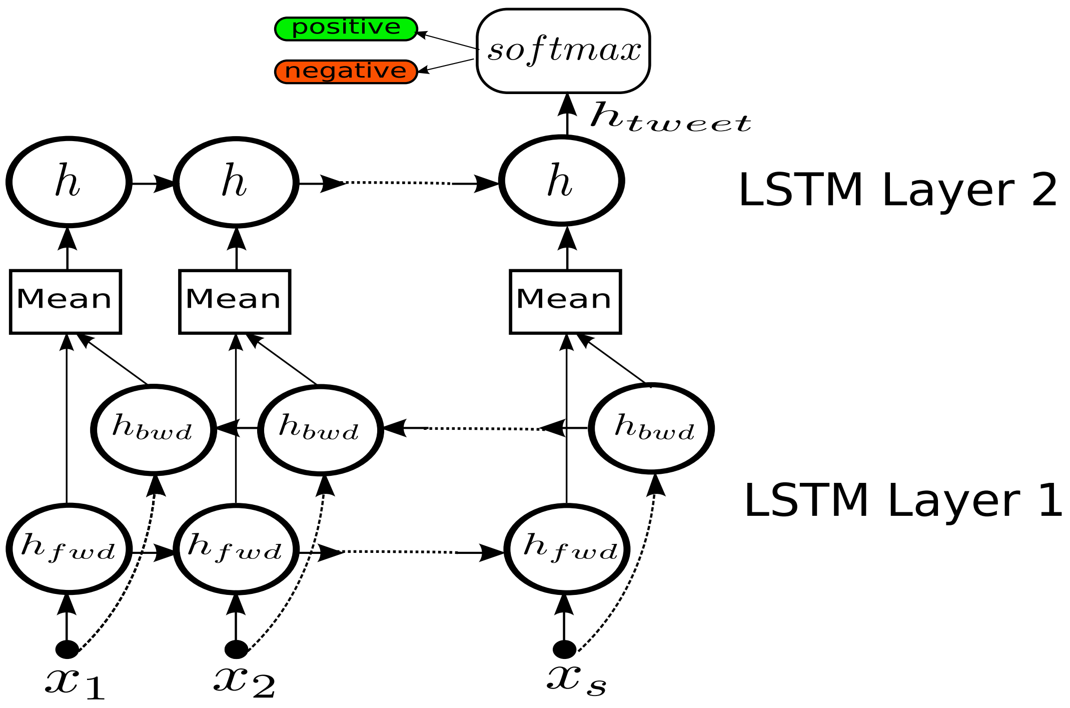

Figure 4: The bi-directional LSTM model used for Twitter sentiment analysis. Each circular node represents an LSTM cell. The input to the model is the sequence x.

the embeddings are always initialized randomly, since pre-trained character embeddings are not yet available.

We explored other variants of the model like have single-directional LSTM, having a single layer as opposed to multiple layers, using concatenation instead of averaging after first layer. We also tried an ensemble LSTM which takes both characters and words as separate inputs which are combined at the last stage by concatenation before feeding to the softmax. Unfortunately, none of these models gave good preliminary results and in the experiments we focused on just the model of Figure 4 with either character or words as inputs.

4 Results

4.1 Datasets

We conduct our experiments on two datasets: the latest benchmark dataset for SemEval 2016 and the dataset provided by [4]. In the latter dataset, the training set consists of 1.6 million weakly-supervised tweets collected during 2009, and the test set is hand-labeled. In the experiments presented below, we first train our models on the Go dataset, then re-train the model parameters on the smaller, fully-supervised SemEval dataset.

Table 1: Data size and label distribution

	Go et. al. (1.6M)		SemEval2016	
	neg	pos	neg	pos
train	800000	800000	781	2805
dev	-	-	358	766
test	177	182	286	886

4.2 Experiments

TODO: verify DCNN results, run on semeval 2016, run using our embeddings?

TODO: add latest ascii/eye results

TODO: evaluate using macro-avg F1, not accuracy

Implementations are available at: `https://github.com/umass-semeval/semeval16`

We compare the performance of character-level models against word-level models. For the former, we compare across character sets ("utf8" and "ascii"), parameter initializations ("rnd" and "eye"), and embedding dimension sizes (50 and 200). The vocabulary for the "utf8" setting consisted of 1949 characters, and for "ascii" consisted of 93 characters. We initialize model parameters randomly at all levels in the "rnd" setting whereas in the "eye" setting, we initialize the second-level LSTM cell and gate parameters using the identity matrix.

We compare word-level models under two settings: initializing the word embeddings using random vectors ("word/rnd") and initialization using "sentiment-specific" word embeddings provided in [23] ("word/sswe").

Our results are shown in Table 2. We train all models using the implementation of the Adam algorithm [12] provided in Lasagne[2] and a learning rate set to 0.1 [3]. To learn word and character embeddings, we use a bidirectional LSTM with 256 hidden units, followed by a mean pooling and a dropout layer ($p = 0.5$), a second forward-directional LSTM (again with 256 hidden units) and a final dropout layer ($p = 0.6$). We obtain final predictions using the softmax function.

Table 2: Accuracy across LSTMs

	1.6M (acc)	semeval (acc)
ascii/rnd/50	83.84	82.08
ascii/rnd/200	82.45	**84.13**
ascii/eye/50	77.44	79.18
utf8	81.34	82.34
char-dcnn	75.0	81.3
word/rnd	81.85	78.07
word/sswe	83.24	79.27
word/dcnn	**87.4**[4]	81.3

The character-level models using the "ascii" character set outperformed the other models on the SemEval dataset. Due to the highly-productive nature of the Twitter "lexicon", users' predisposition toward using slang dialects, and the constraint of the 140-character limit, it makes sense that word-level models underperform.

It is worth noting that the "utf8" model performed comparably despite having a much larger vocabulary size. It is not suprising that the "utf8" model performed worse than the "ascii" model, as the test data consisted of only English tweets; however, since the "utf8" model is implicitly multi-lingual, our results suggest that the same model may perform well across multiple languages. We leave such experiments for future work.

As Table 2 shows, [8] outperform our character LSTM on the Go dataset. However, the results presented for our character and word-level models achieve competitive performance on the Go dataset without tuning hyperparameters such as learning rate and network width.

4.3 Qualitative Analysis

TODO: comparison with word-level model results

Figure 5 shows the effect of character repetition on model confidence over the course of the sequence, where confidence is computed using the softmax function:

[2] `https://github.com/Lasagne/Lasagne`

[3] We retrained the ascii/rnd/200 on SemEval using AdaGrad and a learning rate of 0.01 to achieve 84.13; using Adam and 0.1 learning rate, the result was 83.21

$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^T \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^T \mathbf{w}_k}}$$

In our experiments, we had $K = 2$ corresponding to binary classification between "positive" and "negative" tweets. Sequences ending in periods ("cool.", "coool.") ended up with less-confident scores than tweets not ending in periods. Repeated exclamation points don't increase the model's confidence in the "positive" label as much as might be expected.

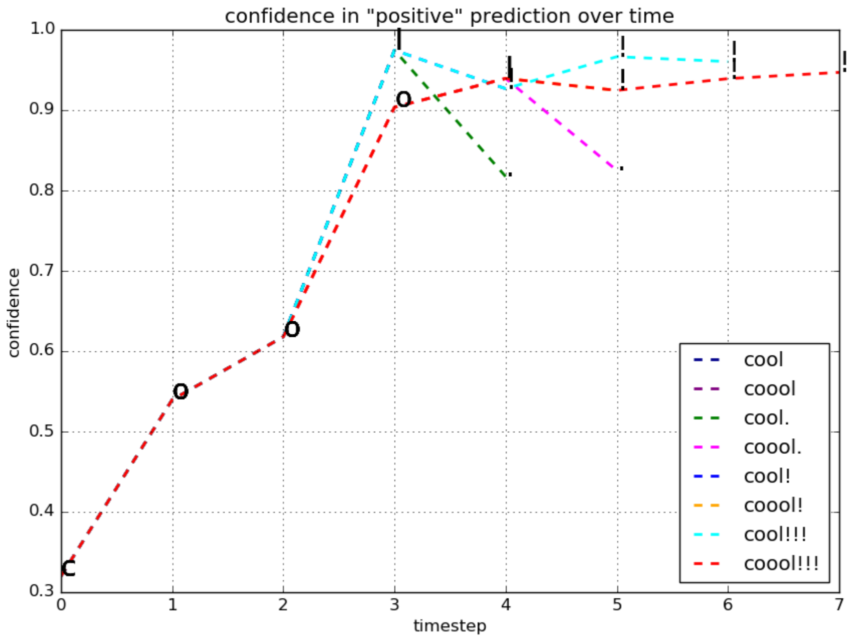

Figure 5: Comparison of model confidence for different forms of the word "cool"

Figure 6 shows that the character-level model learns word meaning at a lexical level: the predictions for "I love puppies" and "I hate puppies" diverges sharply after the model has finished reading "lov" ("love") and "hat" ("hate").

Figure 7 provides further evidence that character-level models can reason about lexical semantics in an intuitive fashion. We compare confidence contours across four tweets from the Go test set. The ground truth labels for the first two tweets are both "negative", and for the second two are both "positive". In the first two, the model finds strong evidence for a "negative" prediction before reaching the word "dentist", and correctly predicts that both tweets are "negative". In the second two, the word "dentist" results in an increase in the model's confidence in a "negative" prediction; however, the words "enjoyable" and ":)" cause the model's confidence to decrease in the "positive" direction.

Figure 8 shows that our model learns the effect of negation. TODO: more details

To further analyze whether related spellings or variants of a word are mapped to nearby points in the final encoding, we fed as input individual words (from the dataset vocabulary) to the character input LSTM model and obtained the encoding of the word after the last layer (before the softmax). We then find the closest neighbors of every word in the encoding space. Table 3 shows words alongside their top nearest neighbors where closeness is measured by cosine similarity. As desired, orthographic variations of the same word "type" are close together in the embedding space. In particular, it is worth noting that "woohoo" with different number of "o" are all close. Similarly, the word "sowwy" which is a common way of saying "sorry" in such informal conversations is also close to the correct word.

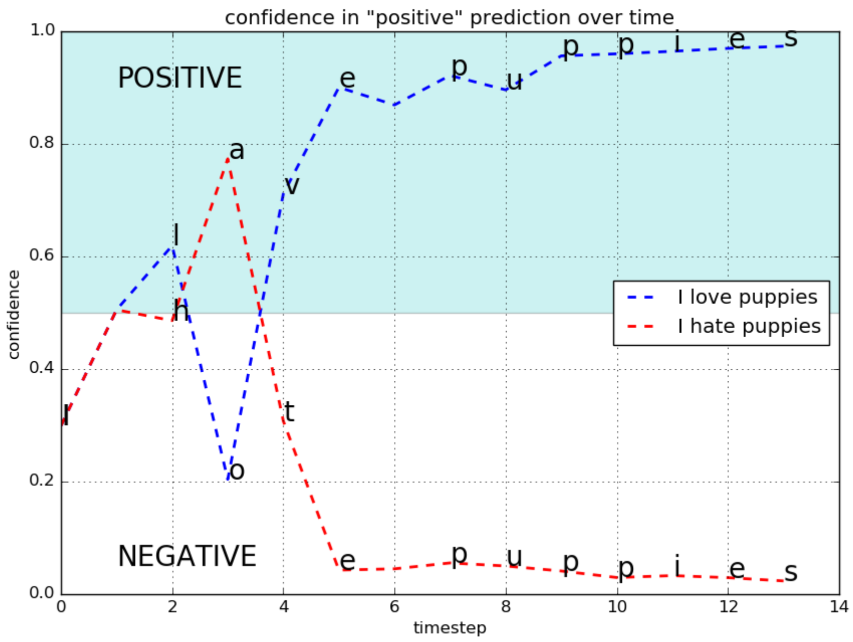

Figure 6: Comparison of model confidence for “I love puppies” vs. “I hate puppies”

Word	Nearest Neighbors
woohoo	wooohoo woohooo wooohooo woohoooo wooooo whoooo woooo whoohoo wohooo
xoxo	xox xoxox zoo xxoo vox xoxoxo zoe xxo twix
thanks	thankss thanku yanks thankx hanks thankz thanksss tanks thanx
sorry	sorrry srry scary sowwy sorryyy scared sory scarred errrr
sucks	sucky suckss yucky eurgh ouchy suckish fucks ouchh jerks
tweet	tweeet tweep tweety tweets textt theee seee outer peace

Table 3: Words and their nearest neighbors

5 Conclusion

In this paper, we show promising empirical evidence that character-level models perform well for sentiment analysis on tweets, which are “noisy”, in that users think of millions of new words and spellings of words every day, and “brief”, in that they are constrained to 140 characters. This type of text data is becoming an important research focus in the field of NLP, as data is often cheap to collect in high volume and can provide important insight into society-wide trends. Though the experiments in this paper focus on sentiment classification, we believe our results provide a basis for future work on character-level modeling for a variety of other NLP tasks.

References

[1] Miguel Ballesteros, Chris Dyer, and Noah A Smith. Improved transition-based parsing by modeling characters instead of words with lstms. *arXiv preprint arXiv:1508.00657*, 2015.

[2] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of Computational Science*, 2(1):1–8, 2011.

[3] Johan Bollen, Alberto Pepe, and Huina Mao. Modeling public mood and emotion: Twitter sentiment and socio-economic phenomena. *arXiv preprint arXiv:0911.1583*, 2009.

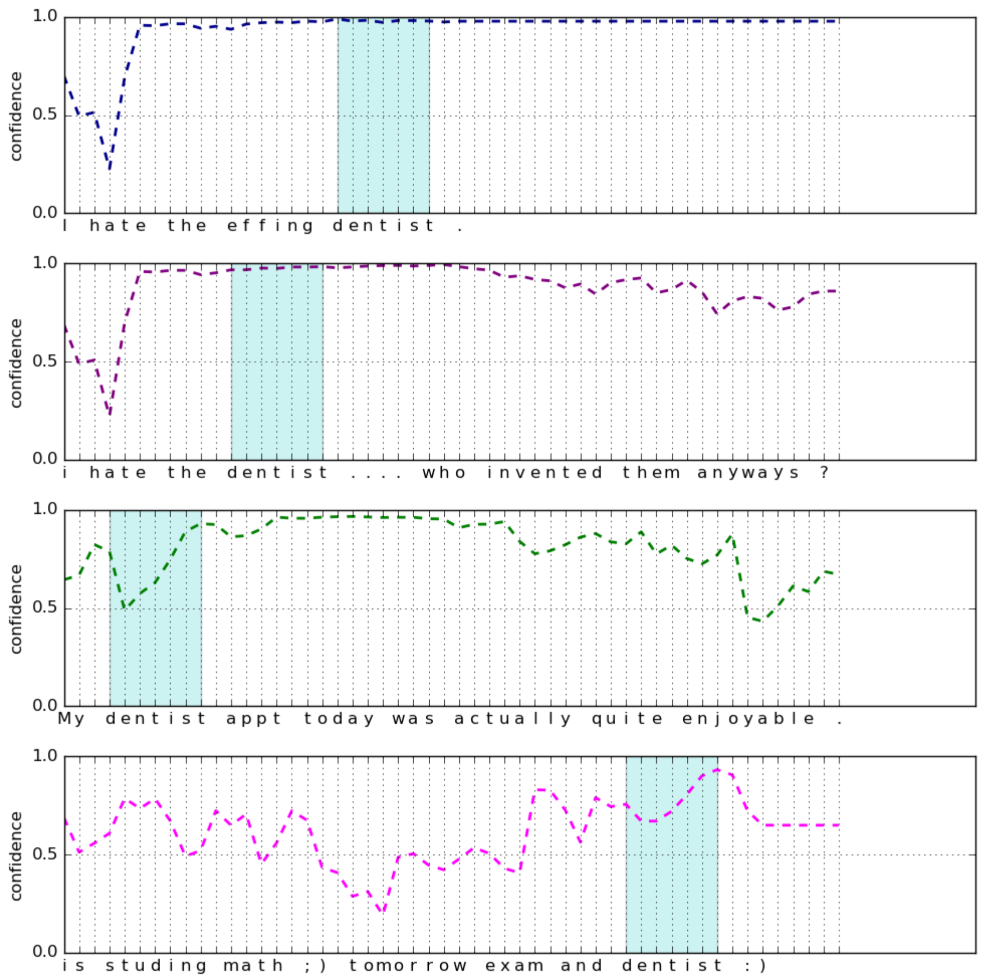

Figure 7: Confidence in "negative" prediction over time across four tweets

[4] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1:12, 2009.

[5] Alex Graves et al. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer, 2012.

[6] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. A field guide to dynamical recurrent neural networks. IEEE Press, 2001.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[8] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[9] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`.

[10] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[11] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*, 2015.

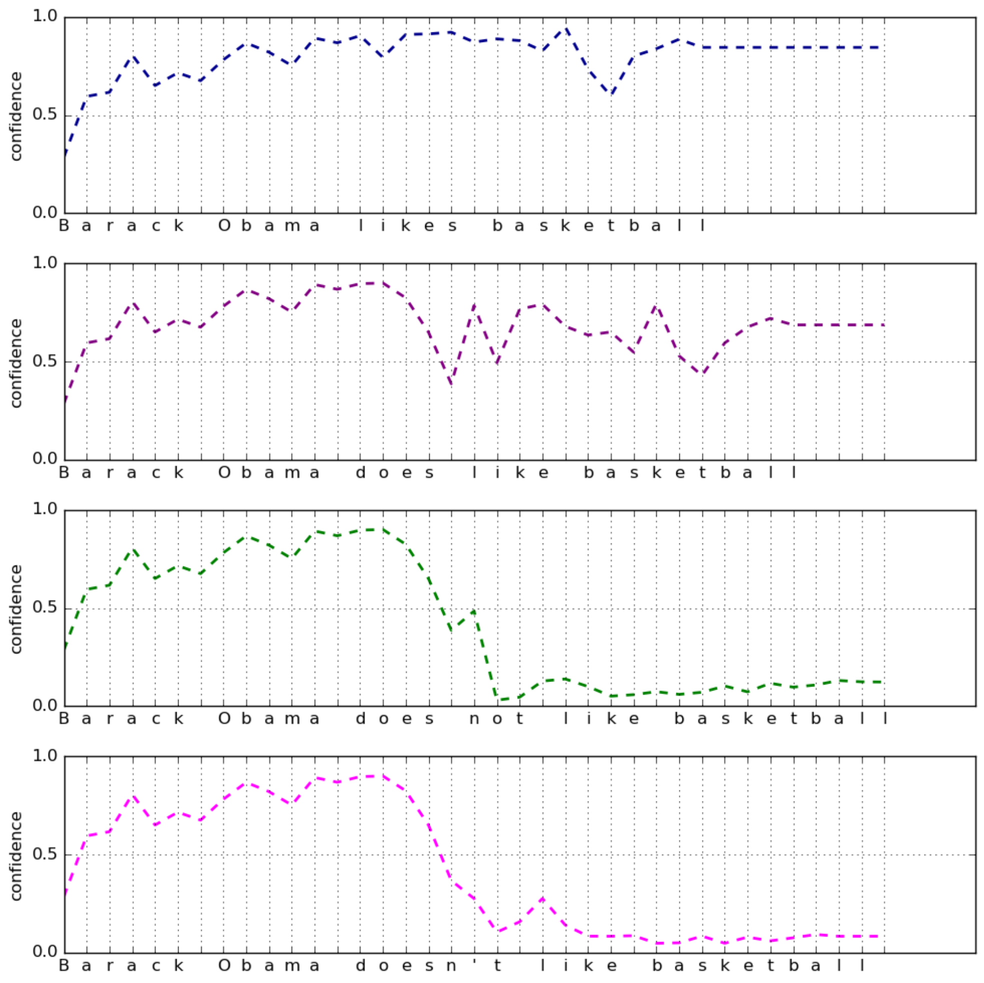

Figure 8: Confidence in "positive" prediction over time across four tweets

[12] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Efthymios Kouloumpis, Theresa Wilson, and Johanna Moore. Twitter sentiment analysis: The good the bad and the omg! *Icwsm*, 11:538–541, 2011.

[14] Jiwei Li, Dan Jurafsky, and Eudard Hovy. When are tree structures necessary for deep learning of representations? *arXiv preprint arXiv:1503.00185*, 2015.

[15] Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. Finding function in form: Compositional character models for open vocabulary word representation. *arXiv preprint arXiv:1508.02096*, 2015.

[16] Wang Ling, Isabel Trancoso, Chris Dyer, and Alan W Black. Character-based neural machine translation. *arXiv preprint arXiv:1511.04586*, 2015.

[17] Rishabh Mehrotra, Scott Sanner, Wray Buntine, and Lexing Xie. Improving lda topic models for microblogs via tweet pooling and automatic labeling. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 889–892. ACM, 2013.

[18] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[19] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[20] Brendan O'Connor, Ramnath Balasubramanyan, Bryan R Routledge, and Noah A Smith. From tweets to polls: Linking text sentiment to public opinion time series. *ICWSM*, 11(122-129):1–2, 2010.

[21] Christopher Olah. Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[22] Aliaksei Severyn and Alessandro Moschitti. Unitn: Training deep convolutional neural network for twitter sentiment classification.

[23] Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 1555–1565, 2014.

[24] Xiaolong Wang, Furu Wei, Xiaohua Liu, Ming Zhou, and Ming Zhang. Topic sentiment analysis in twitter: a graph-based hashtag sentiment classification approach. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1031–1040. ACM, 2011.

[25] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, pages 649–657, 2015.