Tweet2Vec: Learning Tweet Embeddings Using Character-level CNN-LSTM Encoder-Decoder

Soroush Vosoughi*
MIT Media Lab
soroush@mit.edu

Prashanth Vijayaraghavan*
MIT Media Lab
pralav@mit.edu

Deb Roy
MIT Media Lab
dkroy@media.mit.edu

ABSTRACT

We present *Tweet2Vec*, a novel method for generating general-purpose vector representation of tweets. The model learns tweet embeddings using character-level CNN-LSTM encoder-decoder. We trained our model on 3 million, randomly selected English-language tweets. The model was evaluated using two methods: tweet semantic similarity and tweet sentiment categorization, outperforming the previous state-of-the-art in both tasks. The evaluations demonstrate the power of the tweet embeddings generated by our model for various tweet categorization tasks. The vector representations generated by our model are generic, and hence can be applied to a variety of tasks. Though the model presented in this paper is trained on English-language tweets, the method presented can be used to learn tweet embeddings for different languages.

CCS Concepts

•**Information systems** → **Document representation;** •**Computing methodologies** → **Neural networks;** *Information extraction; Lexical semantics;*

Keywords

Twitter; Embedding; Tweet; Convolutional Neural Networks; CNN; LSTM; Tweet2Vec; Encoder-decoder

1. INTRODUCTION

In recent years, the micro-blogging site Twitter has become a major social media platform with hundreds of millions of users. The short (140 character limit), noisy and idiosyncratic nature of tweets make standard information retrieval and data mining methods ill-suited to Twitter. Consequently, there has been an ever growing body of IR and data mining literature focusing on Twitter. However, most of these works employ extensive feature engineering to create task-specific, hand-crafted features. This is time consuming and inefficient as new features need to be engineered for every task.

In this paper, we present *Tweet2Vec*, a method for generating general-purpose vector representation of tweets that can be used for any classification task. *Tweet2Vec* removes the need for expansive feature engineering and can be used to train any standard off-the-shelf classifier (e.g., logistic regression, svm, etc). *Tweet2Vec* uses a CNN-LSTM encoder-decoder model that operates at the character level to learn and generate vector representation of tweets. Our method is especially useful for natural language processing tasks on Twitter where it is particularly difficult to engineer features, such as speech-act classification and stance detection (as shown in our previous works on these topics [13, 12]).

There has been several works on generating embeddings for words, most famously *Word2Vec* by Mikolov et al. [9]). There has also been a number of different works that use encoder-decoder models based on long short-term memory (LSTM) [11], and gated recurrent neural networks (GRU) [1]. These methods have been used mostly in the context of machine translation. The encoder maps the sentence from the source language to a vector representation, while the decoder conditions on this encoded vector for translating it to the target language. Perhaps the work most related to ours is the work of Le and Mikolov [7], where they extended the *Word2Vec* model to generate representations for sentences (called *ParagraphVec*). However, these models all function at the word level, making them ill-suited to the extremely noisy and idiosyncratic nature of tweets. Our character-level model, on the other hand, can better deal with the noise and idiosyncrasies in tweets. We plan to make our model and the data used to train it publicly available to be used by other researchers that work with tweets.

2. CNN-LSTM ENCODER-DECODER

In this section, we describe the CNN-LSTM encoder-decoder model that operates at the character level and generates vector representation of tweets. The encoder consists of convolutional layers to extract features from the characters and an LSTM layer to encode the sequence of features to a vector representation, while the decoder consists of two LSTM layers which predict the character at each time step from the output of encoder.

2.1 Character-Level CNN Tweet Model

Character-level CNN (CharCNN) is a slight variant of the deep character-level convolutional neural network intro-

*The first two authors contributed equally to this work.

SIGIR '16, July 17 - 21, 2016, Pisa, Italy

ISBN 978-1-4503-4069-4/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2911451.2914762

duced by Zhang et al [15]. In this model, we perform temporal convolutional and temporal max-pooling operations, which computes one-dimensional convolution and pooling functions, respectively, between input and output. Given a discrete input function $f(x) \in [1, l] \mapsto \mathbb{R}$, a discrete kernel function $k(x) \in [1, m] \mapsto \mathbb{R}$ and stride s, the convolution $g(y) \in [1, (l - m + 1)/s] \mapsto \mathbb{R}$ between $k(x)$ and $f(x)$ and pooling operation $h(y) \in [1, (l - m + 1)/s] \mapsto \mathbb{R}$ of $f(x)$ is calculated as:

$$g(y) = \sum_{x=1}^{m} k(x) \cdot f(y \cdot s - x + c) \tag{1}$$

$$h(y) = max_{x=1}^{m} f(y \cdot s - x + c) \tag{2}$$

where $c = m - s + 1$ is an offset constant.

We adapted this model, which employs temporal convolution and pooling operations, for tweets. The character set includes the English alphabets, numbers, special characters and unknown character. There are 70 characters in total, given below:

```
abcdefghijklmnopqrstuvwxyz0123456789
-,;.!?:'"/\|_#$%&ˆ*˜‘+-=<>()[]{}
```

Each character in the tweets can be encoded using one-hot vector $x_i \in \{0, 1\}^{70}$. Hence, the tweets are represented as a binary matrix $x_{1..150} \in \{0, 1\}^{150x70}$ with padding wherever necessary, where 150 is the maximum number of characters in a tweet (140 tweet characters and padding) and 70 is the size of the character set.

Each tweet, in the form of a matrix, is now fed into a deep model consisting of four 1-d convolutional layers. A convolution operation employs a filter $w \in \mathbb{R}^l$, to extract n-gram character feature from a sliding window of l characters at the first layer and learns abstract textual features in the subsequent layers. The convolution in the first layer operates on sliding windows of character (size l), and the convolutions in deeper layers are defined in a similar way. Generally, for tweet s, a feature c_i at layer h is generated by:

$$c_i^{(h)}(s) = g(w^{(h)}.\hat{c}_i^{(h-1)} + b^{(h)}) \tag{3}$$

where $\hat{c}_i^{(0)} = x_{i...i+l-1}$, $b^{(h)} \in \mathbb{R}$ is the bias at layer h and g is a rectified linear unit.

This filter w is applied across all possible windows of characters in the tweet to produce a feature map. The output of the convolutional layer is followed by a 1-d max-overtime pooling operation [2] over the feature map and selects the maximum value as the prominent feature from the current filter. In this way, we apply n filters at each layer. Pooling size may vary at each layer (given by $p^{(h)}$ at layer h). The pooling operation shrinks the size of the feature representation and filters out trivial features like unnecessary combination of characters. The window length l, number of filters f, pooling size p at each layer are given in Table 1.

We define $CharCNN(T)$ to denote the character-level CNN operation on input tweet matrix T. The output from the last convolutional layer of CharCNN(T) (size: 10×512) is subsequently given as input to the LSTM layer. Since LSTM works on sequences (explained in Section 2.2 and 2.3), pooling operation is restricted to the first two layers of the model (as shown in Table 1).

Table 1: Layer Parameters of CharCNN

Layer (h)	*Window Size* (l)	*Filters* (f)	*Pool Size* (p)
1	7	512	3
2	7	512	3
3	3	512	N/A
4	3	512	N/A

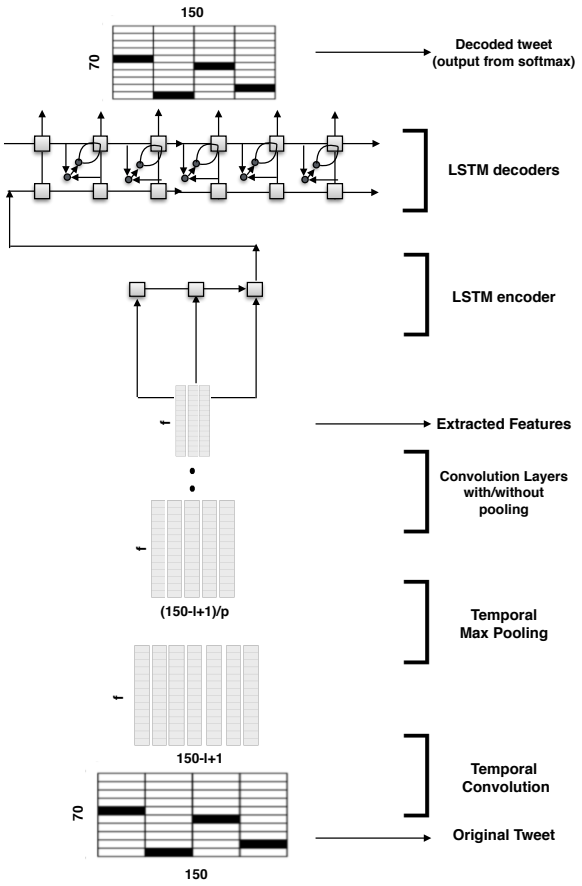

Figure 1: Illustration of the CNN-LSTM Encoder-Decoder Model

2.2 Long-Short Term Memory (LSTM)

In this section we briefly describe the LSTM model [4]. Given an input sequence $X = (x_1, x_2, ..., x_N)$, LSTM computes the hidden vector sequence $h = (h_1, h_2, ..., h_N)$ and and output vector sequence $Y = (y_1, y_2, ..., y_N)$. At each time step, the output of the module is controlled by a set of gates as a function of the previous hidden state h_{t1} and the input at the current time step x_t, the forget gate f_t, the input gate i_t, and the output gate o_t. These gates collectively decide the transitions of the current memory cell c_t and the current hidden state h_t. The LSTM transition functions are defined as follows:

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ l_t &= tanh(W_l \cdot [h_{t-1}, x_t] + b_l) \\ o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ c_t &= f_t \odot c_{t-1} + i_t \odot l_t \\ h_t &= o_t \odot tanh(c_t) \end{aligned} \tag{4}$$

Here, σ is the *sigmoid* function that has an output in [0, 1], *tanh* denotes the hyperbolic tangent function that has an output in $[-1, 1]$, and $\odot$ denotes the component-wise multiplication. The extent to which the information in the old memory cell is discarded is controlled by f_t, while i_t controls the extent to which new information is stored in the current memory cell, and o_t is the output based on the memory cell c_t. LSTM is explicitly designed for learning long-term dependencies, and therefore we choose LSTM after the convolution layer to learn dependencies in the sequence of extracted features. In sequence-to-sequence generation tasks, an LSTM defines a distribution over outputs and sequentially predicts tokens using a softmax function.

$$P(Y|X) = \prod_{t \in [1,N]} \frac{exp(g(h_{t-1}, y_t))}{\sum_{y'} exp(g(h_{t1}, y'_t)} \quad (5)$$

where g is the activation function. For simplicity, we define $LSTM(x_t, h_{t-1})$ to denote the LSTM operation on input x at time-step t and the previous hidden state h_{t-1}.

2.3 The Combined Model

The CNN-LSTM encoder-decoder model draws on the intuition that the sequence of features (e.g. character and word n-grams) extracted from CNN can be encoded into a vector representation using LSTM that can embed the meaning of the whole tweet. Figure 1 illustrates the complete encoder-decoder model. The input and output to the model are the tweet represented as a matrix where each row is the one-hot vector representation of the characters. The procedure for encoding and decoding is explained in the following section.

2.3.1 Encoder

Given a tweet in the matrix form T (size: 150×70), the CNN (Section 2.1) extracts the features from the character representation. The one-dimensional convolution involves a filter vector sliding over a sequence and detecting features at different positions. The new successive higher-order window representations then are fed into LSTM (Section 2.2). Since LSTM extracts representation from sequence input, we will not apply pooling after convolution at the higher layers of Character-level CNN model. The encoding procedure can be summarized as:

$$H^{conv} = CharCNN(T) \quad (6)$$

$$h_t = LSTM(g_t, h_{t-1}) \quad (7)$$

where $g = H^{conv}$ is an extracted feature matrix where each row can be considered as a time-step for the LSTM and h_t is the hidden representation at time-step t. LSTM operates on each row of the H^{conv} along with the hidden vectors from previous time-step to produce embedding for the subsequent time-steps. The vector output at the final time-step, enc_N, is used to represent the entire tweet. In our case, the size of the enc_N is 256.

2.3.2 Decoder

The decoder operates on the encoded representation with two layers of LSTMs. In the initial time-step, the end-to-end output from the encoding procedure is used as the original input into first LSTM layer. The last LSTM decoder generates each character, C, sequentially and combines it with previously generated hidden vectors of size 128, h_{t-1}, for the next time-step prediction. The prediction of character at each time step is given by:

$$P(C_t|\cdot) = softmax(T_t, h_{t-1}) \quad (8)$$

where C_t refers to the character at time-step t, T_t represents the one-hot vector of the character at time-step t. The result from the softmax is a decoded tweet matrix T^{dec}, which is eventually compared with the actual tweet or a synonym-replaced version of the tweet (explained in Section 3) for learning the parameters of the model.

3. DATA AUGMENTATION & TRAINING

We trained the CNN-LSTM encoder-decoder model on 3 million randomly selected English-language tweets populated using data augmentation techniques, which are useful for controlling generalization error for deep learning models. Data augmentation, in our context, refers to replicating tweet and replacing some of the words in the replicated tweets with their synonyms. These synonyms are obtained from WordNet [3] which contains words grouped together on the basis of their meanings. This involves selection of replaceable words (example of non-replaceable words are stopwords, user names, hash tags, etc) from the tweet and the number of words n to be replaced. The probability of the number, n, is given by a geometric distribution with parameter l in which $P[n] \sim l^n$. Words generally have several synonyms, thus the synonym index m, of a given word is also determined by another geometric distribution in which $P[s] \sim r^m$. In our encoder-decoder model, we decode the encoded representation to the actual tweet or a synonym-replaced version of the tweet from the augmented data. We used $p = 0.5$, $r = 0.5$ for our training. We also make sure that the POS tags of the replaced words are not completely different from the actual words. For regularization, we apply a dropout mechanism after the penultimate layer. This prevents co-adaptation of hidden units by randomly setting a proportion ρ of the hidden units to zero (for our case, we set $\rho = 0.5$).

To learn the model parameters, we minimize the cross-entropy loss as the training objective using the Adam Optimization algorithm [5]. It is given by

$$CrossEnt(p, q) = -\sum p(x) \log(q(x)) \quad (9)$$

where p is the true distribution (one-hot vector representing characters in the tweet) and q is the output of the softmax. This, in turn, corresponds to computing the negative log-probability of the true class.

4. EXPERIMENTS

We evaluated our model using two classification tasks: *Tweet semantic relatedness* and *Tweet sentiment classification.*

4.1 Semantic Relatedness

The first evaluation is based on the SemEval 2015-Task 1: *Paraphrase and Semantic Similarity in Twitter* [14]. Given a pair of tweets, the goal is to predict their semantic equivalence (i.e., if they express the same or very similar meaning), through a binary yes/no judgement. The dataset provided for this task contains 18K tweet pairs for training and 1K

pairs for testing, with 35% of these pairs being paraphrases, and 65% non-paraphrases.

We first extract the vector representation of all the tweets in the dataset using our *Tweet2Vec* model. We use two features to represent a tweet pair. Given two tweet vectors r and s, we compute their element-wise product $r \cdot s$ and their absolute difference $|r-s|$ and concatenate them together (Similar to [6]). We then train a logistic regression model on these features using the dataset. Cross-validation is used for tuning the threshold for classification. In contrast to our model, most of the methods used for this task were largely based on extensive use of feature engineering, or a combination of feature engineering with semantic spaces. Table 2 shows the performance of our model compared to the top four models in the SemEval 2015 competition, and also a model that was trained using *ParagraphVec*. Our model (*Tweet2Vec*) outperforms all these models, without resorting to extensive task-specific feature engineering.

Table 2: Results of the paraphrase and semantic similarity in Twitter task.

Model	*Precision*	*Recall*	$F1-Score$
ParagraphVec	0.570	0.680	0.620
nnfeats	**0.767**	0.583	0.662
ikr	0.569	**0.806**	0.667
linearsvm	0.683	0.663	0.672
svckernel	0.680	0.669	0.674
Tweet2Vec	0.679	0.686	**0.677**

4.2 Sentiment Classification

The second evaluation is based on the SemEval 2015-Task 10B: *Twitter Message Polarity Classification* [10]. Given a tweet, the task is to classify it as either positive, negative or neutral in sentiment. The size of the training and test sets were 9,520 tweets and 2,380 tweets respectively (38% positive, 15% negative, and 47% neutral).

As with the last task, we first extract the vector representation of all the tweets in the dataset using *Tweet2Vec* and use that to train a logistic regression classifier using the vector representations. Even though there are three classes, the SemEval task is a binary task. The performance is measured as the average F1-score of the positive and the negative class. Table 3 shows the performance of our model compared to the top four models in the SemEval 2015 competition (note that only the F1-score is reported by SemEval for this task) and *ParagraphVec*. Our model outperforms all these models, again without resorting to any feature engineering.

Table 3: Results of Twitter sentiment classification task.

Model	*Precision*	*Recall*	$F1-Score$
ParagraphVec	0.600	0.680	0.637
INESC-ID	N/A	N/A	0.642
lsislif	N/A	N/A	0.643
unitn	N/A	N/A	0.646
Webis	N/A	N/A	0.648
Tweet2Vec	0.675	0.719	**0.656**

5. CONCLUSION AND FUTURE WORK

In this paper, we presented *Tweet2Vec*, a novel method for generating general-purpose vector representation of tweets, using a character-level CNN-LSTM encoder-decoder architecture. To the best of our knowledge, ours is the first attempt at learning and applying character-level tweet embeddings. Our character-level model can deal with the noisy and peculiar nature of tweets better than methods that generate embeddings at the word level. Our model is also robust to synonyms with the help of our data augmentation technique using WordNet.

The vector representations generated by our model are generic, and thus can be applied to tasks of different nature. We evaluated our model using two different SemEval 2015 tasks: Twitter semantic relatedness, and sentiment classification. Simple, off-the-shelf logistic regression classifiers trained using the vector representations generated by our model outperformed the top-performing methods for both tasks, without the need for any extensive feature engineering. This was despite the fact that due to resource limitations, our *Tweet2Vec* model was trained on a relatively small set (3 million tweets). Also, our method outperformed *ParagraphVec*, which is an extension of *Word2Vec* to handle sentences. This is a small but noteworthy illustration of why our tweet embeddings are best-suited to deal with the noise and idiosyncrasies of tweets.

For future work, we plan to extend the method to include: 1) Augmentation of data through reordering the words in the tweets to make the model robust to word-order, 2) Exploiting attention mechanism [8] in our model to improve alignment of words in tweets during decoding, which could improve the overall performance.

6. REFERENCES

[1] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[2] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[3] C. Fellbaum. *WordNet*. Wiley Online Library, 1998.

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

[5] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[6] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler. Skip-thought vectors. In *Advances in Neural Information Processing Systems*, pages 3276–3284, 2015.

[7] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. *arXiv preprint arXiv:1405.4053*, 2014.

[8] J. Li, M.-T. Luong, and D. Jurafsky. A hierarchical neural autoencoder for paragraphs and documents. *arXiv preprint arXiv:1506.01057*, 2015.

[9] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[10] S. Rosenthal, P. Nakov, S. Kiritchenko, S. Mohammad, A. Ritter, and V. Stoyanov. Semeval-2015 task 10: Sentiment analysis in twitter. In *SemEval*, 2015.

[11] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[12] P. Vijayaraghavan, I. Sysoev, S. Vosoughi, and D. Roy. Deepstance at semeval-2016 task 6: Detecting stance in tweets using character and word-level cnns. 2016.

[13] S. Vosoughi and D. Roy. Tweet acts: A speech act classifier for twitter. In *proceedings of the 10th ICWSM*, 2016.

[14] W. Xu, C. Callison-Burch, and W. B. Dolan. Semeval-2015 task 1: Paraphrase and semantic similarity in twitter (pit). In *SemEval*, 2015.

[15] X. Zhang and Y. LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.