NCSU_SAS_WOOKHEE: A Deep Contextual Long-Short Term Memory Model for Text Normalization

Wookhee Min **Bradford W. Mott**
Center for Educational Informatics
North Carolina State University
Raleigh, NC, USA
{wmin, bwmott}@ncsu.edu

Abstract

To address the challenges of normalizing online conversational texts prevalent in social media, we propose a contextual long-short term memory (LSTM) recurrent neural network based approach, augmented with a self-generated dictionary normalization technique. Our approach utilizes a sequence of characters as well as the part-of-speech associated with words without harnessing any external lexical resources. This work is evaluated on the English Tweet data set provided by the ACL 2015 W-NUT Normalization of Noisy Text shared task. The results, by achieving second place (F1 score: 81.75%) in the constrained track of the competition, indicate that the proposed LSTM-based approach is a promising tool for normalizing non-standard language.

1 Introduction

Recent years have seen increasing use of online social media such as Twitter and Facebook that has generated a growing body of text where non-standard language is prevalent. These non-standard lexical items take many different forms, including unintentional errors based on users' cognitive misconceptions and typographical errors, and intentional non-canonical language such as abbreviations, word lengthening by duplication of characters, Internet slang, phonetic substitutions, and creative use of language (Chrupała, 2014; Owoputi et al., 2013).

A key challenge posed by these non-standard texts is the negative impact on traditional natural language processing (NLP) pipeline processes, evidenced by noticeable underperformance of their predictive accuracy in various domains such as part-of-speech tagging (Gimpel et al., 2011) and named entity recognition (Ritter et al., 2011) compared to more standard text. As an approach to addressing this challenge, text normalization techniques have been widely investigated, ranging from extracting domain specific lexical variants (Gouws et al., 2011), unified letter transformation (Liu et al., 2011), dictionary based methods using string substitution (Han et al., 2012) in an unsupervised manner, to character-level edit operation predictions utilizing conditional random fields in a supervised manner (Chrupała, 2014).

Because language data consists of sequential information, such as streams of characters and sequences of words, many NLP approaches leverage computational models that can effectively deal with temporal data, such as hidden Markov models and conditional random fields (Täckström, 2013; Chrupała, 2014). More recently, deep learning models (e.g., multi-layer feed-forward neural networks, recurrent neural networks, recursive neural networks) have been used in NLP to achieve state-of-the-art performance in areas such as speech recognition (Hinton et al., 2012) and sentiment analysis (Socher et al., 2013). The success of deep learning has been attainable with the emergence of effective training methods for deep networks, such as pre-training (Vincent et al., 2010) and optimization techniques (Zeiler, 2010; Martens and Sutskever, 2011) that significantly diminish problems associated with vanishing and exploding gradient that

Proceedings of the ACL 2015 Workshop on Noisy User-generated Text, pages 111–119,
Beijing, China, July 31, 2015.

are often observed in multi-layer neural network training.

In this work, we leverage long-short term memory models (LSTMs) (Hochreiter and Schmidhuber, 1997; Graves, 2012), a variant of recurrent neural networks, to conduct text normalization on the data set given from the W-NUT English lexical normalization shared task (Baldwin et al., 2015). We additionally harness the part-of-speech tagger created by Noah's Ark research team (Owoputi et al., 2013), a free resource to the constrained task. Similar to Chrupała's work that predicts Levenshtein edit operations between canonical and non-canonical forms of words (Chrupała, 2014), this proposed approach predicts word-level edit operations based on character-level inputs. The proposed approach is novel compared to previous work from four perspectives: (1) it utilizes LSTMs to predict the word-level edit operations, along with a dictionary induced from the training set, (2) it takes as input the surrounding words as well as the current word to capture contextual information of the predicted word, while any additional contextual information (e.g., part-of-speech tags) is treated as heading characters of the word, (3) character and part-of-speech embeddings are learned on the fly in the normalization task instead of having them trained in a separate model, and (4) the self-generated dictionary based normalization as an antecedent step provides statistically significant F1 gains over the standalone computational model.

2 System Architecture

Our proposed system consists of three steps. In the first step, it filters out domain-specific entities such as tokens beginning with @, hash-tags, and URLs. Next, the system searches for words contained in a dictionary generated solely from the training data and normalizes them when appropriate. If words are not normalized in the previous steps, they are passed to the third step, where an LSTM model predicts the canonical form of the word, utilizing the word itself and surrounding words (the previous and following word). In the next sub-sections, we detail how each of the three steps collaboratively operate and explain how the LSTM model is learned based on the training set along with a high-level illustration of the architecture.

2.1 Sequence Flow

The proposed model operates in three primary phases: (1) domain-specific entity filtering, (2) dictionary-based normalization, and (3) LSTM-based normalization.

The first step performs a simple preprocessing of input words. First, every word is converted to the corresponding lowercase word. Second, words that are hash-tags, at-mentions, or URLs are filtered out and left as-is. This preprocessing is useful since the W-NUT normalization task guideline suggests not changing domain-specific entities, and including them could possibly inject noise into predictive models for non-filtered word prediction.

Second, to conduct the dictionary-based normalization, a dictionary is generated from the training set as an index of raw tokens with a list of their normalized forms. For instance, words such as "*ur*" and "*no*" are multiple mapped words, where multiple canonical forms are observed in the training set such as ["*your*", "*you are*"] and ["*no*", "*know*", "*not*"], respectively. This is mainly because they are normalized differently depending on the context used in tweets. On the other hand, words that have a single mapped word are unique in terms of post-normalization form. We denote the first type of words (multiple mapping) as *ambiguous words*, and the second type of words (single mapping) as *unique words*. We use this mapping information as a criterion to decide whether to normalize words based on the dictionary or pass the decision over to the LSTM model in the third step. If a word in the test set turns out to be a unique word, we label the word with the corresponding mapping as defined in the dictionary; otherwise, we pass the word to the LSTM model. This is based on the assumption that these unique words are more likely to have the same unique form in unseen texts.

Finally, once the second normalization step is completed, only words that are either ambiguous or out-of-vocabulary determined by the training set-based dictionary are left and sent to the LSTM model. For ambiguous words, it is important that the model accurately identify the right usage of the words considering context in the tweets. Additionally, it is necessary to capture common patterns of standard words, so that the model can predict canonical forms of words

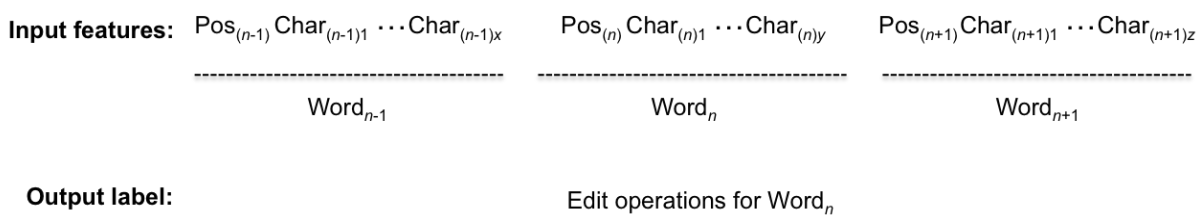

Figure 1: The input features based on characters and POSs and output label (edit operations) used to find the normalized form of *n*th word ($Word_n$). The preceding word ($Word_{n-1}$) and the following word ($Word_{n+1}$)'s POS and constituting characters are combined with the current word ($Word_n$) as input features, where $Word_{n-1}$, $Word_n$, and $Word_{n+1}$ has x, y, and z characters, respectively.

even when they are not seen in the training set. In this work, we formulate an LSTM model to predict an integrated list of edit operations of a word to become a standard word, leveraging its contextual information.

2.2 Data Set Encoding for LSTM

A preliminary step to induce an LSTM model is to encode the data set in a trainable format (i.e., specification of input features and output labels). We define the input format as a list of sequential, lowercased characters that compose the previous word, current word, and following word. Each character is mapped to a unique index (0–66), since there are a total of 67 different characters in the training data after the preprocessing step described in 2.1. If the current word does not have a previous or next word (e.g., the first or last word in a tweet), a padding character is assigned for the previous or following word to have a consistent format. In this work, we additionally consider a word's part-of-speech (POS) as extra input to the model, as previous literature (e.g., Yarowsky, 1997; Täckström, 2013) indicates POS tags can improve performance in other natural language processing tasks, such as text-to-speech synthesis and NLP parsers. We use an off-the-shelf POS tagger that features Brown clustering: the CMU Twitter Part-of-Speech Tagger, which achieves a state-of-the-art tagging result of 93% on a Twitter benchmark data set (Owoputi et al., 2013). The extracted POS information is added as a distinct heading character to each word, so that they are leveraged in the LSTM models. Similar to the character padding, we apply a POS padding for missing previous or next words. Note that leveraging POSs about words is extendable to utilize any other meta-information, and we examine the feasibility of applying this technique with POS tags in the context of text normalization. The input encoding for predicting edit operations of the current word ($Word_n$) is described in Figure 1. To summarize, the number of inputs in a sequence is $3 + x + y + z$, where x, y and z are the number of characters of the previous, current, and following word, respectively, while 3 is derived from the POS tags of all three words.

Encoding the output is based on the Levenshtein distance algorithm (Levenshtein, 1966) that supports three operations: *insert*, *replace*, and *delete*, inspired by Chrupała's text normalization work (Chrupała, 2014). In this work, we reformulate his approach to predict word-level edit operations instead of character-level edit operations, by which the model predicts a label for an individual token. In the character-level prediction, to correctly normalize a token, it requires all correction predictions on every character that belongs to a word (i.e., probabilities get multiplied), whereas the word-level prediction requires one prediction per token.

Once a training sample is given, the Levenshtein distance algorithm calculates the required edit operations to convert the possibly non-standard word into the corresponding canonical form. For example, "*dese*" and "*dey*" with the canonical forms of "these" and "they", respectively, have edit operations of "*insert_t_replace_h*, *none*, *none*, *none*, *none*" and "*insert_t_replace_h*, *none*, *none*, *none*" (a comma is used as a delimiter for characters). Note that the insert operation only supports inserting a character before the current character, so to support insertions at the end of a word, every input word (e.g., "*doin*") is concatenated with an empty character (e.g., "*doin* "), and edit operations are applied on the empty character (e.g., "*none*, *none*, *none*, *none*, *insert_g*"). Since more class labels make this multi-class classification task more challenging, it is important to have an optimized set of edit operation labels, while the model should be still capable of converting to canonical words based on the given edit opera-

tion. For the preceding example, a way to shrink the label size is omitting repeated *none* operations at the end of the string. With this optimization applied, both "*dese*" and "*dey*" have the same edit operation of "*insert_t_replace_h*" ignoring all following "*none*"s. From this, the model can successfully decode the operation by replacing the first character of "*d*" with "*th*" and appending following stream of characters from each example, thereby constructing "*these*" and "*they*", respectively. A more pronounced benefit of this technique can be found when there is no change required in terms of the edit operations. No change examples will have a single common label of *nothing*; otherwise, a series of *none* operations will be generated as independent labels based on the length of the word.

Another challenge in the edit operation approach lies in dealing with variants of repeated occurrence of the same character (e.g., "*sooo*", "*soooo*") often used to emphasize the word, since all required edit operations will be treated as different labels (e.g., "*none*, *none*, *delete*, *delete*", "*none*, *none*, *delete*, *delete*, *delete*") in spite of similar forms of edits (note that we omit the last *none* due to the previously-mentioned optimization). To address this challenge, we attempt to replace characters that subsequently occur more than two times with a single character or double characters, and see if the converted word exists in the dictionary (in this work, the double character conversions have a higher priority over the single character conversion, if both exist in the dictionary). If it appears in the dictionary, we use the word as an input word and calculate the edit operations based on the converted word setting; otherwise, we use the original word as the input word. We expect this would reduce the number of possible labels (e.g., both "*sooo*" and "*soooo*" are converted to "*so*", as "*so*" is defined as a canonical form in the dictionary while "*soo*" is not, and so both of them have edit operations of nothing). As a result, the total number of labels obtained form the training set is reduced from 706 to 694.

Training examples for LSTMs are built upon all words except for hash-tags, at-mentions, and URLs that are filtered in the first step, regardless of whether a word is ambiguous or unique. In this manner, we expect that LSTMs can capture context information from every three-word example and thus utilize all available contextual dependencies when ambiguous or out-of-vocabulary words appear in the test set.

2.3 Long-Short Term Memory (LSTM) for Text Normalization

An LSTM (Hochreiter and Schmidhuber, 1997) is a variant of recurrent neural networks (RNNs) that is specifically designed for sequence labeling on temporal data. LSTM has been extended to have a longer term memory compared to traditional RNNs by introducing a memory block that features one or more self-connected memory cells along with three gating units: input gate, forget gate and output gate (Graves, 2012). Traditional RNNs often suffer from vanishing and exploding gradient problems when training deep networks using the backpropagation-through-time method, and thus prevent RNNs from storing long-term dependencies from previous time steps in the sequential data. In LSTMs, the input and output gate modulate the incoming and outgoing signals on the memory cell, and the forget gate controls the previous state of the memory cell whether to remember or forget; this structure allows it to preserve gradient information over long periods of time, and thus effectively address vanishing/exploding gradients that make training difficult in standard RNNs (Graves, 2012).

We use an LSTM as our base model, as described in Figure 2. Note that in the figure, the previous word (Word_{t-1}) and the following word (Word_{t+1}) are omitted due to the space limitations; it is important to note that they have the same structure as in the current word (Word_t). For a word's edit operation prediction, three words and their associated POSs are fed into the model in the form of a sequence of characters for each word. As noted above, the POS of each word is inserted before the first character of that word, regarded as a heading character that provides extra information for the associated word.

When a deep learning model takes words or characters as input, an approach to obtaining their representation is using *one-hot-encoding*, which is a bit vector whose length is the size of the vocabulary of words or characters, where only the associated word/character bit is on (i.e., 1) while all other bits are off (i.e., 0). Another popular approach is utilizing word/character embeddings, where their representations are learned in the context of unsupervised language modeling (Mikolov et al., 2013; Pennington et al., 2014) or supervised tasks of interest (Mesnil et al., 2013). We choose the latter approach and learn character embeddings using a linear projection layer while training the text normalization LSTM model in a supervised manner. We set

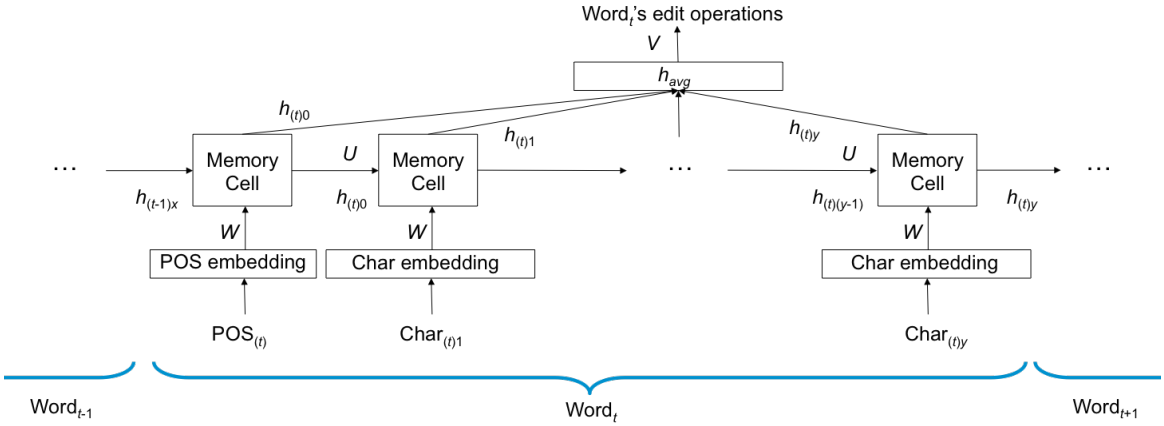

Figure 2: An illustration of the LSTM-based text normalization model

both the character and POS embedding size to 256 for this task based on preliminary analyses using a grid search.

For our base code, we utilized a Theano-based (Bastien et al., 2012) LSTM implementation[1] with a single-cell memory block per time, which was implemented targeting a sentiment analysis task on an IMDB data set.

In this implementation, the input gate (i_t), forget gate (f_t), and candidate value of the memory content ($\widetilde{c_t}$) at time *t* are computed by Equation (1), (2), and (3), respectively, in which *W* and *U* are weight matrices for the input (x_t) at time *t* and the cell output (h_{t-1}) at time *t*-1, *b* is the bias vector of each unit, and σ and tanh are the logistic sigmoid and hyperbolic tangent function, respectively:

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (1)$$
$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \quad (2)$$
$$\widetilde{c_t} = \tanh(W_c x_t + U_c h_{t-1} + b_c) \quad (3)$$

Once the three vectors are computed, the current memory cell's state is updated to a new state (c_t), by modulating the current memory candidate value ($\widetilde{c_t}$) via the input gate (i_t) and the previous memory cell state (c_{t-1}) via the forget gate (f_t). Through this process, a memory cell decides whether to keep or forget the previous memory state and regulates the candidate of the current memory state via the input gate. This step is described in Equation (4):

$$c_t = i_t \widetilde{c_t} + f_t c_{t-1} \quad (4)$$

In Equation (5), the output gate (o_t), similarly calculated as in Equation (1) and (2), is utilized to compute the cell activation (h_t) of the LSTM block, based on the new memory state (c_t) (Equation 6):

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \quad (5)$$
$$h_t = o_t \tanh(c_t) \quad (6)$$

In this model, as a variant of the LSTM proposed by Graves (2012), the input and forget gates do not take as input the previous memory cell's state, and the output gate does not utilize the current memory cell's state, to take advantage of a computational benefit when training models; rather, the current memory cell's state is only utilized to calculate the cell's output representation, along with the computed vector from the output gate (Equation 6).

As illustrated in Figure 2, a character or POS is fed into the model at each time step, inducing a cell output (*h*) and a cell state (*c*). To predict the label (i.e., edit operations of a word), the model performs an average pooling (h_{avg}) on a sequence of computed cell output representations ($h_{(t-1)0}$ to $h_{(t+1)z}$) on the training example with the three word input sequence, calculates posterior probabilities of all candidate labels using the averaged representation (h_{avg}) in a softmax layer, and chooses the label with the highest posterior probability value as prediction.

[1] http://deeplearning.net/tutorial/lstm.html

	Without Dictionary Normalization			With Dictionary Normalization		
	Precision	Recall	F1	Precision	Recall	F1
Fold 1	0.8777	0.6735	0.7622	0.8803	0.7185	0.7912
Fold 2	0.9036	0.6546	0.7592	0.9134	0.7232	0.8072
Fold 3	0.8737	0.6352	0.7356	0.8797	0.6805	0.7674
Fold 4	0.8671	0.6501	0.7431	0.9107	0.6986	0.7907
Fold 5	0.8388	0.6867	0.7551	0.8859	0.7347	0.8032
Averaged score	**0.8722**	**0.6600**	**0.7510**	**0.8940**	**0.7111**	**0.7919**

Table 1: 5-fold cross validation results of LSTMs without dictionary normalization and with dictionary normalization.

	Non-contextual Model			Contextual Model		
	Precision	Recall	F1	Precision	Recall	F1
Fold 1	0.9032	0.6838	0.7783	0.8803	0.7185	0.7912
Fold 2	0.8776	0.7419	0.8041	0.9134	0.7232	0.8072
Fold 3	0.8988	0.6704	0.7680	0.8797	0.6805	0.7674
Fold 4	0.9209	0.6961	0.7929	0.9107	0.6986	0.7907
Fold 5	0.8589	0.7387	0.7943	0.8859	0.7347	0.8032
Averaged score	**0.8919**	**0.7062**	**0.7875**	**0.8940**	**0.7111**	**0.7919**

Table 2: 5-fold cross validation results of LSTMs: non-contextual model vs. contextual model.

For other parameter settings in this experiment, we used 256 hidden units, 25% dropout rate (Srivastava et al., 2014), ADADELTA (Zeiler, 2012) for the network optimization, negative log-likelihood for the cost function, and mini-batch based gradient descent with the batch size set to 16. To avoid overfitting, we set aside a separate validation set, and let the training process repeat until there is no progress within the last ten iterations in terms of performance on the validation set.

3 Empirical Evaluation

Before submitting our test set result to the W-NUT English lexical normalization shared task, we ran a 5-fold cross validation on the training set to evaluate the proposed approach. To conduct the experiment, we split the training set into 5 partitions based on a Tweet-level separation, and trained an LSTM model, iteratively using 4 out of the 5 partitions in each fold.

In the first evaluation, we examine two variations of our approach to measure the impact of dictionary-based normalization as an intermediate step: (1) applying phase 1 and phase 3, in which we do not leverage dictionary-based normalization but predict labels based on an LSTM model after at-mention and hash-tag and URL filtering, and (2) applying all three phases. The evaluation is conducted on contextual models that take three word inputs.

Table 1 describes the result of these two approaches for each fold. For pairwise comparison of the two approaches, we conduct a Wilcoxon signed-rank test on F1 rates. The result indicates that there is a statistically significant improvement in F1 rates (79.19% by achieving 5.4% marginal improvement) for "with dictionary normalization" over "without dictionary normalization" (Z=-2.023, p=0.043). To examine the effects of the LSTM-based model, we further evaluated a without-LSTM approach (phase 1 and 2 only), in which all out-of-vocabulary words are left unchanged, and the most often observed canonical form in the dictionary is used as the label for ambiguous words (if the frequency is tied, the first form in the hash table is used). The average F1 score of this dictionary only model is 0.7786; the LSTM

model with the dictionary statistically outperforms the dictionary only model (Z=-2.023, p=0.043).

In the second evaluation, we additionally compare another set of two variations: contextual model (taking surrounding words as well as the current word) vs. non-contextual model (only taking the current word). Table 2 summarizes the comparison on the two approaches enriched with the dictionary normalization. The contextual model outperforms the non-contextual model in terms of the F1 score, but the difference does not elicit a statistically significant difference (Z=-1.214, p=0.225).

To construct a final model for the test set prediction, we utilize an ensemble method on contextual LSTM models with dictionary normalization. Given a test set, we calculate the prediction probability from each of the 5 models induced from the five-fold cross validation, multiply the probability values from the softmax layer, and choose the label with the highest resulting probability. In the evaluation through the W-NUT competition, this approach (NCSU_SAS_WOOKHEE.cm) achieved a precision score of 91.36%, recall score of 73.98%, and F1 score of 81.75%, placing second in the constrained text normalization track.

4 Conclusion and Future Work

Text normalization is a key capability for addressing the challenges posed by noisy text. This paper presents a contextual long-short term memory based normalization method, augmented with a dictionary-based normalization technique. Evaluations with the training set indicate that the dictionary-based normalization significantly outperforms the without-dictionary model. This method was evaluated on the English Tweet test set offered by the W-NUT shared task, and shows promise as a lexical normalizer for noisy texts by achieving an F1 score of 81.75%. We conclude that inputs encoded with a sequence of characters are a natural fit for the LSTM's temporal structure when normalizing non-standard language.

In the future, it will be important to investigate if including more surrounding words as context contributes to the model's performance, and examine possibilities of using different types of word-level meta-data as additional heading characters in the model. Another direction for future work is to investigate adaptations of the LSTM model with a self-generated dictionary. For example, when a word is an ambiguous word, the LSTM's prediction is not necessarily part of the normalization candidates given by the dictionary for the word. A tight coupling between the LSTM model and the candidate list or building a separate model targeted to only ambiguous words may significantly increase performance.

References

Timothy Baldwin, Marie Catherine de Marneffe, Bo Han, Young-Bum Kim, Alan Ritter, and Wei Xu. 2015. Shared Tasks of the 2015 Workshop on Noisy User-generated Text: Twitter Lexical Normalization and Named Entity Recognition. In *Proceedings of the Workshop on Noisy User-generated Text (WNUT 2015)*, Beijing, China.

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. 2012. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*.

Grzegorz Chrupała. Normalizing tweets with edit scripts and recurrent neural embeddings. 2014. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 680–686. Association for Computational Linguistics.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. 2011. Part-of-speech tagging for Twitter: Annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 42–47. Association for Computational Linguistics.

Stephan Gouws, Dirk Hovy, and Donald Metzler. 2011. Unsupervised mining of lexical variants from noisy text. In *Proceedings of the First workshop on Unsupervised Learning in NLP*, pages 82–90. Association for Computational Linguistics.

Alex Graves. 2012. *Supervised sequence labelling with recurrent neural networks*. Heidelberg: Springer.

Bo Han, Paul Cook, and Timothy Baldwin. 2012. Automatically constructing a normalisation dictionary for microblogs. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 421–432. Association for Computational Linguistics.

Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior et al. 2012. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6): 82–97.

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

V. I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady*, 10(8): 707–710.

Fei Liu, Fuliang Weng, Bingqing Wang, and Yang Liu. 2011. Insertion, deletion, or substitution?: normalizing text messages without pre-categorization nor supervision. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 71–76. Association for Computational Linguistics.

James Martens and Ilya Sutskever. 2011. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040.

Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. 2013. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Proceedings of the 14th Annual Conference of the International Speech Communication Association*, pages 3771–3775.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A. Smith. 2013. Improved Part-of-Speech Tagging for Online Conversational Text with Word Clusters. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–390.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. 2014. In *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*.

Alan Ritter, Sam Clark, and Oren Etzioni. Named entity recognition in tweets: an experimental study. 2011. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pp. 1524–1534. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Stroudsburg, PA, October. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15 (1): 1929–1958.

Oscar Täckström, Dipanjan Das, Slav Petrov, Ryan McDonald, and Joakim Nivre. 2013. Token and type constraints for cross-lingual part-of-speech tagging. *Transactions of the Association for Computational Linguistics*, 1: 1–12.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion.

The Journal of Machine Learning Research, 11: 3371–3408.

David Yarowsky. 1997. Homograph disambiguation in text-to-speech synthesis. In *Progress in speech synthesis*, pages 157–172. Springer, New York.

Matthew D. Zeiler. 2012. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*