Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference

Deep Neural Network Architecture for Character-Level Learning on Short Text

Joseph D. Prusa, Taghi M. Khoshgoftaar
jprusa@fau.edu, khoshgof@fau.edu
Florida Atlantic University

Abstract

Character-level deep learning for text classification tasks enables models to be trained without any prior knowledge of the data or language; however, an optimal neural network design for different text domains is not known and may vary. In this paper, we expand on current efforts to train neural networks from character-level data by conducting an experimental investigation on neural network design for text classification of short text documents. We trained and evaluated four networks, two consisting of convolutional layers followed by dense layers and two consisting of convolutional layers followed by a LSTM layer. Our experimental results show tweets need network architectures compatible with their short length. Networks found effective for other sentiment classification tasks may not produce an effective classifier in this domain, if their architecture is ill-suited for short instances.

Introduction

Given sufficient data, the use of deep neural networks has been demonstrated to be an effective approach for a wide variety of machine learning tasks, including many text learning tasks where large datasets are available. Deep learning methods are popular for feature extraction as they can find relationships between words and/or phrases allowing a condensed feature space of abstracted data representations to be generated. Googles word2vec is an example of this as it provides an automated means of extracting semantic representations from big data. Using a large-scale text corpus, or other sufficiently large datasets, word2vec constructs a vocabulary of a fixed size and learns how to describe words outside the vocabulary by constructing vector representations using words from within the vocabulary (Najafabadi et al. 2015).

Deep learning can also be used for end-to-end discriminative tasks such as text classification. Deep learning text classifiers learn relationships between basic text features, such as words, and class labels so that computers can discriminate between concepts encountered in text. A common starting point for this process is word-level and morphological features; however, there is an inherent loss of information with these data representations and a dependence on human domain knowledge. Recent efforts have started to investigate character-level learning (Zhang and LeCun 2015; Prusa and Khoshgoftaar 2016; Xiao and Cho 2016). In this new paradigm, models are trained from raw text data with no prior knowledge of the domain or language. Thus, the existence of words, parts of speech and other grammatical constructs are learned from scratch. This removes any potential for human bias in feature engineering and ensures that all information from the text is available for training.

While character-level learning has been demonstrated effective for a variety of text classification tasks, many network blueprints currently exist and the best architecture may depend on the specific classification task and dataset. Two popular types of networks are Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), such as the Long Short Term Memory (LSTM) network. Each type of network has its merits, and they can also be combined in a single network. Both types of networks and their combination have been used with word-level learning for a wide variety classification tasks; however, character-level learning is less developed.

Our work seeks to expand research on character-level learning by exploring different neural network designs on short text data. We evaluate four networks, two networks consisting of convolutional layers followed by dense layers and two consisting of convolutional layers followed by a LSTM layer on tweet sentiment data. To the best of our knowledge, this is the first paper to train character-level networks combining convolutional and recurrent layers on tweet sentiment data. Additionally, we demonstrate network architectures found effective for specific text classification tasks may not work for dissimilar text. Specifically, text length and the semantic nature of documents is important when designing neural networks, as a classifier found effective for movie and restaurant review sentiment may not work for tweet sentiment. This indicates there is no best network, even for closely related text. Due to space limitations, testing of additional network designs and training on additional datasets is left for future work.

The following section presents related works on deep learning for text classification. This is followed by a explanation of how convolutional and LSTM networks work, our experimental design, results, and finally our conclusions.

Copyright © 2017, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Related Works

Two common deep neural network architectures for performing text classification are Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). CNNs gained popularity for their effectiveness in computer vision and image classification and have been used for end-to-end discriminative text classification tasks involving the identification of high-level concepts prevalent throughout an entire document. Additionally, CNNs have been demonstrated to be effective for natural language processing tasks such as part-of-speech tagging, named entity extraction, semantic roles, and semantically similar words (Collobert and Weston 2008).

Kim (Kim 2014), demonstrated training a CNN on top of static, pre-made word vectors performs well for sentence classification tasks. They showed a network with a single convolutional layer, followed by dense layers, outperformed non deep learning algorithms, such as Multinomial Naïve Bayes and Support Vector Machine (SVM), on several benchmarking datasets. However, other studies show using CNNs to build high-level features from word-level representations, then using these features with conventional classifiers performs better than end-to-end neural network classification. One such study was conducted by Poria et al. (Poria, Cambria, and Gelbukh 2015). They used a CNN to build high-level features from textual data, represented by a 306 dimensional vector consisting of a word vector and part of speech values. The output of the penultimate fully connected layer (the last layer before the classification layer) was then used to generate features for use with other algorithms. They found using these features with SVM produced a better performing classification model than relying on the CNNs final output layer for classification.

RNNs are designed for learning tasks with sequential data, making them a logical choice for text classification tasks as text is sequential and prior words or characters can shape the meaning of subsequent words or characters. A popular version of the RNNs, the LSTM network has been demonstrated to be effective for a wide range of text classification tasks. Despite producing models with high performance, LSTMs are also slow to train and prone to overfitting (Graves 2012). Additionally, they are sensitive to network architecture changes and hyper-parameter tuning. Thus, training a LSTM network can be considerably more challenging than using a CNN, potentially leading to poorer performance and longer training times. In the domain of sentiment classification, CNNs have been found to outperform RNNs such as LSTM networks (Xu, Liang, and Baldwin 2016). The issue of training time is further compounded by the lack of an RNN library equivalent to NVIDIAs CuDNN library (Chetlur et al. 2014).

RNNs can also be used in conjunction with other types of networks to improve performance. Tang et al. (Tang, Qin, and Liu 2015) investigated adding a Gated Recurrent Neural Network (GRNN) after either convolutional layers (CNN-GRNN) or an LSTM layer (LSTM-GRNN). They trained models on Yelp and IMDB sentiment data and compared them against SVM models with several different feature sets (constructed from word-level data representations) and a CNN. They found both the CNN-GRNN and LSTM-GRNN outperformed their baseline approaches, indicating that these new models are a better choice for document-level sentiment classification than previous approaches.

Character-Level Deep Learning

While the majority of these studies have worked from word-level data representations such as bag-of-word or word2vec models, the use of character-level data representations has become more common in recent works. Zhang and LeCun (Zhang and LeCun 2015) first proposed the use of character-level data representations for training a CNN classifier by representing text as a sequence of character vectors, forming an image-like data matrix for each text instance. They showed they could learn high-level text concepts and beat state-of-the-art models by training a neural network from character data with no prior feature engineering or extraction, or any knowledge of language such as the existence of words or parts of speech. Their approach employed 1-of-m embedding to represent characters. Each character is then represented by a vector of size m, where m is the number of characters in their alphabet. Each instance is then a sequence of these character vectors. Using this embedding, they trained deep convolutional neural networks on a variety of text classification benchmarking datasets and found models using character-level data outperformed the use of features generated with bag-of-words, bag-of-centroids and the deep learning approach word2vec for many of their datasets.

Prusa and Khoshgoftaar (Prusa and Khoshgoftaar 2016) also investigated character-level learning. They proposed a new character embedding to create a more compact data representation, reducing training time and memory requirements. Similar to 1-of-m embedding, they constructed vector representations of characters; however, they allowed multiple entries in each vector to be non-zero. Thus a vector of length $\log(m)$ could be used in place of a vector of length m. They evaluated their embedding using CNNs trained from tweet sentiment data and found that their embedding resulted in significantly better classification performance than using 1-of-m in addition to requiring less memory and having the advantage of faster training.

Kim et al. (Kim et al. 2015) demonstrated LSTM networks can effectively learn from character-level data, and like CNNs models learning from character-level data, can outperform networks trained from word and morphological features. Additionally, their character-level models have fewer parameters than their word-level counterparts since an alphabet of characters if far smaller than even a limited vocabulary of words. Xiao and Cho (Xiao and Cho 2016) tested substituting the multiple dense layers in a CNN network with an LSTM layer followed by the decision layer. Additionally, they employed an embedding layer to create dense character vectors from character-level data in a manner similar to word2vec. They trained and tested four CNN-LSTM models with two to five convolutional layers and one LSTM layer against networks with six convolutional layers and three dense layers, as used in (Zhang and LeCun 2015), on multiple datasets with paragraph to page length documents. They found the CNN-LSTM networks had higher

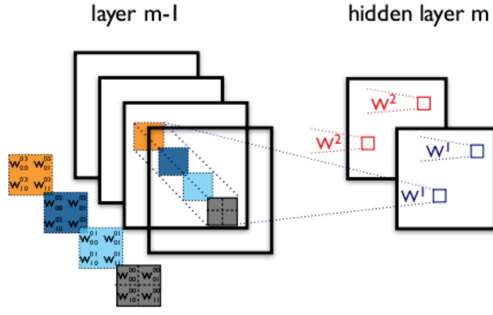

Figure 1: Visualization of output from CNN Filters.

performance for the majority of tested datasets; however, there was no best number of convolutional layers prior to the LSTM layer.

Deep Neural Networks

In addition to densely connected layers similar to those found in a multilayer perceptron network, we train networks with convolutional layers and networks with LSTM layers.

Convolutional Layers

A convolutional layer is a sparsely connected layer that only connects neurons in subsequent layers to a small region in the current input volume. A feature map is obtained by convolving a linear filter consisting of weights and a bias term across the layer's input space, then applying a non-linear activation function. Multiple randomly initialized filters are employed in each layer forming a set of feature maps, $\{\mathbf{h}^k, k = 0..K\}$, where K is the number of feature maps in the layer. Figure 1 provides a visualization. Thus, the kth feature map, $\mathbf{h}^k$, with filter weight $\mathbf{W}^k$, bias term $\mathbf{b}_k$ and non-linear activation function ϕ for input vector $\mathbf{x}_{ij}$, can be defined as:

$$\mathbf{h}_{ij}^k = \phi(\mathbf{W}^k \mathbf{x}_{ij} + \mathbf{b}_k)$$

Common receptor field sizes are small, typically 3×3, or 5×5 for two dimensional inputs and length 3 or 5 for 1 dimensional inputs. Thus, for character-level text learning, an individual layer only learns local relationships between a character and its closest neighbors; however, by stacking multiple convolutional layers, hierarchical relationships can be learned spanning larger portions of text. Convolutional layers have a number of hyperparameters including filter size, number of filters, convolutional stride and choice of activation function. We elect to use Rectified Linear Units (ReLU), defined as $f(x) = \mathbf{max}(0, x)$, as our activation function as it promotes nonlinear responses (Nair and Hinton 2010), may be more biological plausible than tanh, and is used in prior works with character-level learning (Zhang and LeCun 2015; Xiao and Cho 2016).

An optional pooling layer may be added after the activation function layer to down-sample the output in an effort to reduce number of network parameters, and avoid overfitting via establishing translation invariance of features. The most commonly used form of pooling is max-pooling, a nonlinear form of down-sampling that only passes on the element with the highest activation from its pool.

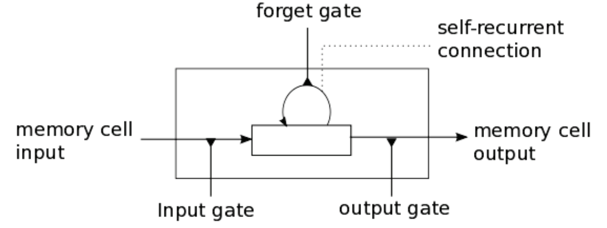

Figure 2: Visualization of a LSTM memory cell and gates.

LSTM Layers

A recurrent layer uses a recursive function, which returns the current hidden state $\mathbf{h}_t$ using an input vector $\mathbf{x}$ and the previous hidden state $\mathbf{h}_{t-1}$. Choosing a simple recursive function, such as $\mathbf{h_t} = \mathbf{tanh}(\mathbf{W_x x_t} + \mathbf{U_x h_{t-1}})$, here $\mathbf{W}_x$ and $\mathbf{U}_x$ are weight matrices, may cause the gradient signal to explode if the weight matrix contains many large values, or vanish if it contains many small values since the gradient may be multiplied by the weight matrix many times in during gradient back propagation.

The Long Short Term Memory (LSTM) network was introduced as a solution to this. The LSTM initially consisted a memory cell composed of an input gate, output gate and candidate memory cell (Hochreiter and Schmidhuber 1997); however, an update was made to the memory cell structure by adding a forget gate (Gers, Schmidhuber, and Cummins 2000). Figure 2 shows a modern memory cell. The exploding/vanishing gradient problem is addressed by including a self-recurrent connection with a weight of 1.0, ensuring the state of a memory cell remains constant without outside interference.

At a timestep t, the input gate $\mathbf{i}_t$, forget gate $\mathbf{f}_t$, candidate memory cell $\widetilde{\mathbf{C}}_t$, output gate $\mathbf{o}_t$, new memory cell $\mathbf{C}_t$, and new hidden state $\mathbf{h}_t$ can be calculated as follows:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i)$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f)$$

$$\widetilde{\mathbf{C}}_t = tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c)$$

$$\mathbf{C}_t = \mathbf{i}_t * \widetilde{\mathbf{C}}_t + \mathbf{f}_t * \mathbf{C}_{t-1}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{V}_o \mathbf{C}_t + \mathbf{b}_o)$$

$$\mathbf{h}_t = \mathbf{o}_t * tanh(\mathbf{C}_t)$$

Where $\mathbf{x}_t$ is the input to the memory cell at time t, $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{V}$ are weight matrices, $\mathbf{b}$ are the bias vectors and σ the sigmoid function $S(t) = \frac{1}{1+e^{-t}}$.

Empirical Design

Character Embedding

We employ two methods of character embeddings to create our data representation. The first is referred to as log(m) embedding and a more compact alternative to 1-of-m embedding. Log(m) embedding represents each character as a vector of length log(m) where m is the alphabet size. This embedding can potentially improve classification performance and also reduced training time and memory requirements compared to 1-of-m embedding (Prusa and Khoshgoftaar 2016). This embedding choice has the advantage of being performed as a part of data preprocessing, thus generates an embedding layer. We select the commonly used 1-Hot character embedding in conjunction with an embedding layer in the network as a second embedding approach. 1-Hot embedding represents each instance as a vector of numeric values (0-255 for the UTF-8 alphabet). This representation is then sent to the network's embedding layer which turns positive integers into dense vectors of fixed size. We use the embedding layer from the Keras library (Chollet 2016).

Dataset

As a source of short text data, we use instances from the sentiment140 corpus (Go, Bhayani, and Huang 2009) for training, validation and test data. The corpus contains 1.6 million instances, equally positive and negative labeled, and was generated by collecting and labeling tweets using emoticons. Tweets are limited to 140 characters; however, when using UTF-8 encoding the length may be longer due to using multiple characters to represent letters and symbols outside UTF-8 such as the Cyrillic alphabet. Due to this, the longest tweet was found to have a length of 374 characters instead of 140. Thus, we padded the length of all instances to 374 characters as our neural networks require input with uniform size. This results in an input of 1x374 for networks using 1-Hot embedding, and an input of 8x374 for our network using the character embedding from (Prusa and Khoshgoftaar 2016). No pre-processing was used.

The 1.6 million instances of the sentiment140 tweet corpus were partitioned into three parts for training, validation and performing a final evaluation of our networks. An initial split was performed where 10% were randomly selected and set aside as test data. The test data is not used in any part of the training process and is used to perform a final performance evaluation. The remaining 90% of the data was then randomly split into training and validation partitions, with 20% (18% of the total data) set aside for validation of each epoch so that an early stopping criteria for training the network could be established. The remaining training data is used to train the parameters of the network.

Network Architecture

In this study, we investigate four neural networks, two networks composed of convolutional layers followed by densely connected layers and two networks composed of convolutional layers followed by a long short term memory layer. The two convolutional plus dense layer networks are based off the network architecture from (Prusa and Khoshgoftaar 2016). The first network, CNN3-2D, uses the 2D character embedding and consists of three convolutional layers followed by a max-pooling layer, then three dense layers. The second network, CNN3-1D, uses 1-Hot character embedding with an embedding layer, followed by a 1D equivalent to the network architecture used in CNN-2D. Both mixed convolutional LSTM networks use 1-Hot character embedding with an embedding layer. The first of these networks, CNN3-LSTM-1D, is a reconstruction designed to replicate the model from (Xiao and Cho 2016). It consists of three convolutional layers followed by an LSTM layer with 128 memory cells, and finally a dense classification layer with two nodes. Max-pooling layers are placed after every convolutional layer. The last network, CNN1-LSTM-1D has a similar architecture, but only one convolutional layer. See Table 1 for further details on network hyper-parameters.

Training and evaluation

Convolutional networks were trained using stochastic gradient descent with Nesterov momentum to match prior experimentation (Zhang and LeCun 2015; Prusa and Khoshgoftaar 2016). The learning rate was set to 0.01 and momentum to 0.9. Convolutional plus LSTM networks were trained with AdaDelta using $\rho = 0.95$ and $\epsilon = 10^{-5}$ to match experimentation prior (Xiao and Cho 2016). During training we evaluate performance of the network by measuring training loss, training accuracy, validation loss and validation accuracy. We used binary cross entropy as our loss function. For target t and output o, it is defined by:

$$loss(t, o) = -t \log(o) - (1 - t) \log(1 - o)$$

Networks were set to train for a maximum of 50 epochs, with early termination of training when no improvement in validation loss was observed for the previous 5 epochs. The random seed for splitting data into training, test and validation was fixed for all experiments so all networks were trained with the same data partitions. Training time per epoch was measured in addition to total training time for all epochs. After training, a final evaluation of each network was performed using accuracy, the number of correctly classified instances divided by the total number of instances, as our performance metric. This is an appropriate metric as the data is balanced and both classes are of equal importance.

Experiments were conducted on a laptop node with 4 Intel Xeon Cores, 64 GB of RAM and a NVIDIA Quadro M5000 GPU accelerator running Windows 10. Networks were constructed using Theano 0.8.0 (Al-Rfou et al. 2016) with NVIDIA CUDA 7.5 (Nickolls et al. 2008), Keras and the NVIDIA CuDNNv5 library (Chetlur et al. 2014).

Results and Discussion

Accuracy, evaluated on the test set, is presented in Table 2. Additionally, average run time per training epoch, the number of epochs before training was terminated and total training time are presented. Best results for each metric are in **bold**. CNN3-LSTM performed very poorly. Due

Network	Embedding	Filter Size	No. Filters	Pooling	LSTM	Dense	Parameters
CNN3-2D	no	3x3, 3x3, 3x3	128	2x2	no	1024, 1024, 2	25,466,370
CNN3-1D	yes	3x1, 3x1, 3x1	128	2x1	no	1024, 1024, 2	1,551,634
CNN3-LSTM	yes	3x1, 3x1, 3x1	128	2x1, 2x1, 2x1	yes	2	367,442
CNN1-LSTM	yes	3x1	128	2x1	yes	2	401,170

Table 1: Network architecture, hyper-parameters and number of parameters (network size).

Model	Accuracy	Time per Epoch	Number of Epochs	Total Time
CNN3-2D	81.07	973.9s	**11**	**10712.9s**
CNN3-1D	**82.24**	**591.9s**	25	14797.5s
CNN3-LSTM-1D	50.18	332.6s	7	2328.2s
CNN1-LSTM-1D	79.45	3788.8s	23	87142.4s

Table 2: Test accuracy, training time per epoch, number of epochs and total training time of each network.

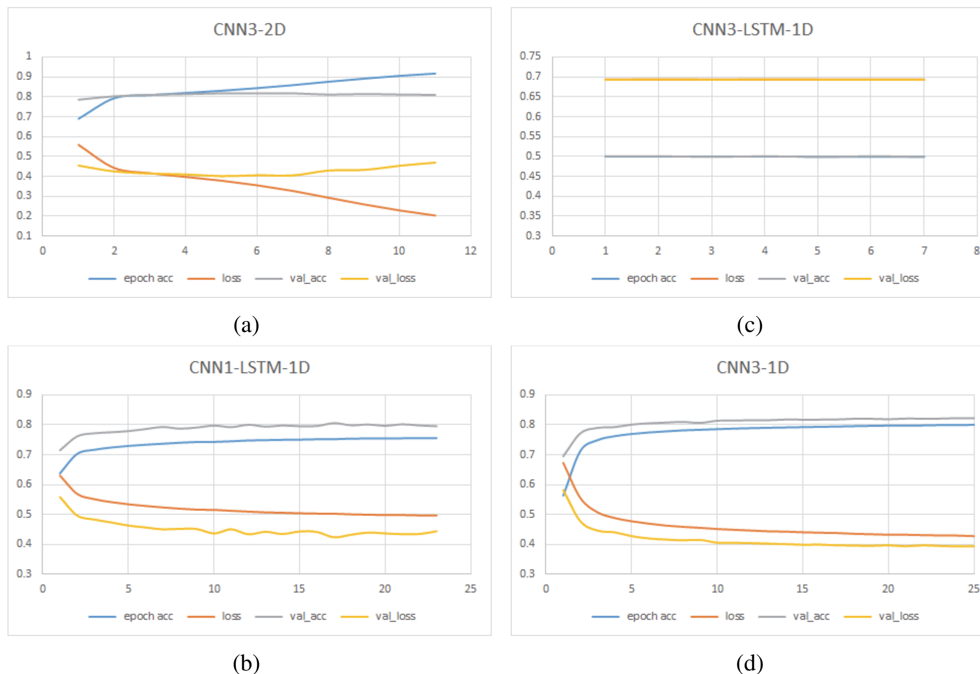

Figure 3: Loss and accuracy for training and validation of Neural Networks

to this failure, additional training attempts to train an effective classifier were conducted to no avail, with tweaks including the elimination of the first two max-pooling layers. Due to this, the training time per epoch and total training time for CNN3-LSTM should not be compared to the other three networks. Both CNN3-2D and CNN3-1D outperform the CNN1-LSTM-1D with our data.

Training loss (loss), training accuracy (epoch_acc), validation loss (val_loss) and validation accuracy (val_acc) are plotted in Figure 3. From Figure 3a, it can be seen that CNN3-2D reaches its minimal validation loss with the fewest number of epochs. Figure 3c confirms that CNN3-LSTM is not effective for our data as both training and validation loss never change.

The poor performance of CNN3-LSTM-1D indicates that the network is ill suited for our data since the other three networks perform well. This network design and the selected hyper-parameters were found to be effective in other text domains (Xiao and Cho 2016). This raises the question, what makes our data different and why would a model that is effective for IMDB sentiment and Yelp Sentiment completely fail when trained on tweet sentiment data. It has previously been speculated (Xiao and Cho 2016) that extra max-pooling layers may negatively impact performance; however, their removal made no impact in preliminary experimentation. Thus, the most likely explanation is that due to the short nature of tweets, learning long sequences of characters is not advantageous and may actually be detrimental as entire tweets might be learned instead of words and phrases leading to overfitting.

An interesting observation is that CNN3-2D and CNN3-1D have similar training times (CNN3-2D is only 50% slower per epoch), despite CNN3-2D being roughly 16 times larger (based on number of parameters in the network) as

seen in Table 1. This is likely due to its use of 2D convolutional layers as opposed to 1D layers, since currently CuDNN library offer rapidly accelerated training of 2D convolutional layers. Since CNN3-2D has faster training (relative to the number of parameters) and requires less training epochs, it finishes training before CNN3-1D. While this network has a slightly lower accuracy on our test set, stopping training earlier (between epochs 5 and 7) may result in higher accuracy in addition to even shorter training time. Due to this, it is currently unclear which network should be considered better for our data.

Compared to either CNN model, each training epoch of CNN1-LSTM-1D takes a very long time. It also has lower classification accuracy than using convolutional layers followed by dense layers. This does not match earlier observations on for text classification with longer text instances. Thus, our current observations support the idea that there is no best NN design for a given text classification task; however, we recommend using convolutional layers followed by dense layers for classifying short text.

Conclusion

In this paper, we provide a case study on neural network design for character-level text classification using tweet data to train sentiment classifiers. We trained and evaluated four networks, two consisting of convolutional layers followed by dense layers and two consisting of convolutional layers followed by a LSTM layer. We found that both architectures can train effective classifiers; however, one of our networks was unable to be trained to effectively classify tweet sentiment. We also found that while replacing dense layers with a LSTM layer reduced the number of parameters in the network, it significantly increases training time. Additionally, using 2D convolutions is much faster than using 1D convolutions due to CuDNN allowing for very fast 2D convolutions to be performed.

Our observations support the use of a network comprised of convolutional layers followed by dense layers for classification of short text, whereas the addition of a LSTM layer has previously been found to be beneficial for text with longer instances (Xiao and Cho 2016). Future work should explore additional networks with more layers and repeat our experiments on more datasets to see if our observations generalize to other short instance text data.

Acknowledgment: We acknowledge partial support by the NSF (CNS-1427536). Opinions, findings, conclusions, or recommendations in this material are the authors and do not reflect the views of the NSF.

References

Al-Rfou, R.; Alain, G.; Almahairi, A.; Angermueller, C.; Bahdanau, D.; Ballas, N.; Bastien, F.; Bayer, J.; Belikov, A.; Belopolsky, A.; et al. 2016. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*.

Chetlur, S.; Woolley, C.; Vandermersch, P.; Cohen, J.; Tran, J.; Catanzaro, B.; and Shelhamer, E. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*.

Chollet, F. 2016. keras. https://github.com/fchollet/keras.

Collobert, R., and Weston, J. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, 160–167. ACM.

Gers, F. A.; Schmidhuber, J.; and Cummins, F. 2000. Learning to forget: Continual prediction with lstm. *Neural computation* 12(10):2451–2471.

Go, A.; Bhayani, R.; and Huang, L. 2009. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford* 1–12.

Graves, A. 2012. Neural networks. In *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer. 15–35.

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

Kim, Y.; Jernite, Y.; Sontag, D.; and Rush, A. M. 2015. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*.

Kim, Y. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Nair, V., and Hinton, G. E. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 807–814.

Najafabadi, M. M.; Villanustre, F.; Khoshgoftaar, T. M.; Seliya, N.; Wald, R.; and Muharemagic, E. 2015. Deep learning applications and challenges in big data analytics. *Journal of Big Data* 2(1):1–21.

Nickolls, J.; Buck, I.; Garland, M.; and Skadron, K. 2008. Scalable parallel programming with cuda. *Queue* 6(2):40–53.

Poria, S.; Cambria, E.; and Gelbukh, A. 2015. Deep convolutional neural network textual features and multiple kernel learning for utterance-level multimodal sentiment analysis. In *Proceedings of EMNLP*, 2539–2544.

Prusa, J. D., and Khoshgoftaar, T. M. 2016. Designing a better data representation for deep neural networks and text classification. In *Information Reuse and Integration (IRI), 2016 IEEE International Conference on*, 411–416.

Tang, D.; Qin, B.; and Liu, T. 2015. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1422–1432.

Xiao, Y., and Cho, K. 2016. Efficient character-level document classification by combining convolution and recurrent layers. *arXiv preprint arXiv:1602.00367*.

Xu, X.; Liang, H.; and Baldwin, T. 2016. Unimelb at semeval-2016 tasks 4a and 4b: An ensemble of neural networks and a word2vec based model for sentiment classification.

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