Convolutions Are All You Need (For Classifying Character Sequences)

Zach Wood-Doughty, Nicholas Andrews, Mark Dredze
Center for Language and Speech Processing
Johns Hopkins University, Baltimore, MD 21218
{zach,noa,mdredze}@cs.jhu.edu

Abstract

While recurrent neural networks (RNNs) are widely used for text classification, they demonstrate poor performance and slow convergence when trained on long sequences. When text is modeled as characters instead of words, the longer sequences make RNNs a poor choice. Convolutional neural networks (CNNs), although somewhat less ubiquitous than RNNs, have an internal structure more appropriate for long-distance character dependencies. To better understand how CNNs and RNNs differ in handling long sequences, we use them for text classification tasks in several character-level social media datasets. The CNN models vastly outperform the RNN models in our experiments, suggesting that CNNs are superior to RNNs at learning to classify character-level data.

1 Text Classification with Sequences

Deep learning has transformed text classification tasks by providing models that can fully account for word order, whereas previous methods required simplifications such as treating documents as a "bag of words." Recurrent neural networks (RNNs) are attractive for their ability to handle variable-length sequences and have contributed huge improvements to machine translation (Bahdanau et al., 2015; Cho et al., 2014) and semantic modeling (Tai et al., 2015; Socher et al., 2013), among many other areas.

Despite this widespread success, RNNs often perform poorly on long sequences – common in document classification – in which the model must learn representations than span many timesteps. If two informative tokens are far apart in a document, a training gradient must maintain information about one such token while being backpropagated through the sequence of per-token learned representations. Formulations like Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) use gating mechanisms designed to prevent such long-distance gradients from vanishing (Hochreiter et al., 2001; Bengio et al., 1994) by allowing constant error flow through the network; yet empirical results find that LSTMs fail to learn long-range dependencies.[1]

Convolutional Neural Networks (CNNs) differ from RNNs in their internal structure, which may make them more promising for modeling long sequences. Whereas RNNs construct a chain of one hidden state for every input token, convolutional models can connect input tokens with paths sublinear in the input sequence's length. CNNs have succeeded at text classification (Kim, 2014; Zhang et al., 2015) and language modeling (Kim et al., 2016). ByteNet, introduced by Kalchbrenner et al. (2016), used dilated convolutions to capture long-range dependencies in character-level machine translation and achieve fast training times. Despite these promising results, prior work has not highlighted specific tasks or domains in which CNNs are expected to outperform RNNs.

We consider the task of classifying social media posts; such user-generated text data contains many unique words through misspellings, lexical variation, and slang. Because a word-level approach requires either an immense vocabulary or a large proportion of out-of-vocabulary tokens (Luong et al., 2014), we model the text one character at a time. This choice allows models to generalize across misspellings ("hello" vs. "helo") or phonetic or emphasized spelling ("hellloo"). The downside of character-level data is the dramatic increase in sequence length, which forces the model to learn longer dependencies. In several character-level datasets containing informal text,

[1]See Section 3.3 of Jozefowicz et al. (2016) and Section 4 of Sundermeyer et al. (2012) for two such results.

Dataset	Number Instances	Number Labels	Char Vocab	Max Length
SST	9.6k	2	97	160
SemEval '17	62k	3	106	144
Yelp	120k	5	168	768
LID utf-8	43k	43	1365	128
LID Bytes	43k	43	198	288

Table 1: The datasets used for character based sequence classification evaluations.

we show CNNs vastly outperform RNNs while training several times faster.

2 Data

We consider the task of sequence classification, a form of document classification where a sequence model is used to produce a single label for a piece of text. Following work that has demonstrated the advantage of character based sequence models for informal text (Vosoughi et al., 2016), we treat the text as a sequence of characters. We consider datasets formed from Twitter posts or single sentences, since these are lengthy enough to force the model to learn long-range dependencies, yet short enough to train our RNN models quickly enough on a single GPU. We set a maximum length (divisible by 16) for each dataset, so as to make the longest sequences more manageable while maintaining most of the variability in lengths. We truncate sequences longer than the maximum length and pad shorter sequences with a unique token. Table 1 summarizes our datasets.

SST Movie Reviews: The Stanford Sentiment Treebank (SST) dataset (Socher et al., 2013) contains 9.6k sentences extracted from a corpus of `rottentomatoes.com` movie reviews, labeled for sentiment by crowdsourced annotators. We use the binary-classification version of the dataset, discarding the 3-star neutral reviews. We look only at entire sentences, and do not use any of the shorter-phrase annotations or parse tree data from the treebank. We use the published splits from the dataset's authors. The vocabulary size is 96 characters. The average and median sequence lengths were 100 and 103 characters. We truncated all sequences at 160 characters, roughly the 90th percentile of all lengths.

SemEval 2017: The task of Twitter sentiment tagging from SemEval 2017 (Task 4, Subtask A) provides 62k English tweets labeled for positive, neutral, or negative sentiment (Rosenthal et al., 2017). The training data is a compilation of all previous SemEval datasets. We use the 4k tweets from the 2013 competition's test data as our development set. We use the provided train and test splits: 46k training and 12k test examples. We preprocess by converting URLs and '@-mentions' into special tokens. The training data has a vocabulary of 106 characters. The average and median sequence lengths were 103 and 110 characters. We truncated all sequences at 144 characters, roughly the 95th percentile.

Yelp Reviews: The 2015 Yelp Dataset Challenge[2] provides a dataset of 4.7M reviews of restaurants and businesses which contain text and a one- to five-star rating. We randomly sample 120k reviews from the dataset and use 100k for training, and 10k each for development and test sets. We limit the character vocabulary to the 168 characters that appear at least 10 times. The average and median sequence lengths were 613 and 437 characters. We truncated all sequences at 768 characters, roughly the 75th percentile.

Twitter LID: Twitter provides a multilingual dataset[3] for language identification (LID) of tweets (Bergsma et al., 2012). We used the recall-focused dataset but were only able to download a subset of the original tweets and so limited our experiments to the 43 languages for which we could download at least 1000 tweets. We preprocessed this dataset in two ways: using the utf-8 encoding and using the raw byte strings. For the utf-8 data, we limit the vocabulary to the 1365 characters that appear at least 10 times. The average and median sequence lengths were 69 and 64 characters. We truncated utf-8 sequences at 128 characters, roughly the 95 percentile of lengths. For the raw bytes data, we use the entire 198 character vocabulary. The average and median sequence lengths were 118 and 102 characters. We truncated byte sequences at 288 characters, roughly the 95th percentile.

3 Models

We consider several recurrent and convolutional models for character based sequence classification tasks. For each model, we feed the learned representation through two 256-dimensional

[2]https://www.yelp.com/dataset/challenge

[3]https://blog.twitter.com/2015/evaluating-language-identification-performance

fully-connected layers to produce an output distribution over labels.

3.1 Recurrent Neural Networks

We consider both Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) and Gated Recurrent Unit (GRU) networks (Chung et al., 2014), two variants of RNNs that use gating to mitigate vanishing gradients. In contrast to LSTMs, GRUs do not have an output gate and thus have fewer parameters. For both types of RNNs, we experimented with unidirectional and bidirectional models, and stacking up to a depth of three layers, with hidden dimensions between 128 to 512. This gives us model sizes from 250k to 14M parameters.

Attention mechanisms, which learn a weighted average of the hidden states, have become popular for sequence-to-sequence tasks (Bahdanau et al., 2015). However, in our classification setting attention means a weighted average across hidden dimension. As a simpler attention baseline we use max-pooling to reduce across the time dimension.

3.2 Convolutional Neural Networks

Recent interest in CNNs for NLP has resulted in new model architectures. We explore several of these in our comparison. Additionally, initial experiments with average-pooling, and other techniques to reduce convolutional outputs across the time dimension, found that global max-pooling worked best for all CNNs.

CNN-A We consider an extremely simple CNN model which is just a stack of convolutional layers. The model can be implemented in a few lines of Tensorflow code. For hyperparameters, we considered filter widths of 1, 2, and 3, with either 128 or 256 filters per convolution, and model depths from 1 to 4 layers, with and without layer normalization, and with and without relu activations. Following (Yu and Koltun, 2015), we double the convolutional dilation at each subsequent layer. This gives us model sizes ranging from 20k to 800k parameters.

CNN-B This is a popular CNN model introduced by Kim (2014). At each layer of the network, the model concatenates the outputs of multiple convolutions with different filter widths. Although this has been more widely used in text classification tasks, it is still quite a simple model to implement. We considered one- and two-layer models, with local max-pooling between the layers. The minimum filter width was two and the maximum was either three or five. We used either 128 or 256 filters per layer. This gives us model sizes ranging from 100k to 1.8M parameters.

ByteNet Kalchbrenner et al. (2016) introduced a convolutional model for character-level machine translation, using dilated convolutions (Yu and Koltun, 2015) and residual multiplicative blocks (Kalchbrenner et al., 2017). This is a much more complicated convolutional model. Their experiments demonstrate state-of-the-art performance on character-level language modeling with a Wikipedia dataset. We vary the number of multiplicative blocks from two to three and the number of dilated convolutions per block from three to five. We used a filter width of two and either 128 or 256 filters per convolution. This gives us model sizes ranging from 200k to 6M parameters.

3.3 N-gram Baseline

As an alternative to our neural sequence models, we also compare against character n-gram models which do not take token order into account. We train these with the `sklearn` SGDClassifier using logistic loss and L2 regularization (Pedregosa et al., 2011). We train models on character 3-, 4-, and 5-grams, and consider regularization parameters in the range from 0.0005 and 5. We pick the n-gram size and regularization parameter on the dev set before evaluating on the test set.

3.4 Model Training

We use cross-entropy loss as our training objective for every model. Initial experiments led us to fix the learning rate at $1e{-}4$ and to use the Adam optimizer (Kingma and Ba, 2014) with beta parameters of 0.85 and 0.997. We embedded character inputs before feeding them into the CNN or RNN models, learning 512-dimensional embeddings for the LID utf-8 dataset and 64-dimensional embeddings for all other datasets.

We trained each model for 200 epochs, with early stopping based on a search for a stable plateau in held-out dev accuracy. Specifically, for each three-epoch window we calculate the minimum dev accuracy in that window. Our results in Table 2 report the test accuracy from the

Dataset	SST	SemEval '17	Yelp	LID utf-8	LID Bytes
N-grams	79.8	59.5	64.6	88.5	90.6
RNN (GRU)	57.9	49.0	60.1	52.1	28.1
CNN-A	72.3	57.5	64.8	72.1	73.2
CNN-B	**78.6**	**59.2**	**65.3**	85.1	**85.0**
ByteNet	66.4	53.7	63.1	**85.6**	84.6
Past work	83.1[a]	66.4[b]	≈67.6[c]	≈92[d]	≈92[d]

Table 2: Test accuracy for each model architecture, with comparisons to n-gram models and past work. The best RNN or CNN result is bolded. Past work: [a]Barnes et al. (2017) compares several models, achieving the best result with a word-level LSTM. [b]Yin et al. (2017) achieved the shared task's highest accuracy using a recurrent CNN (Lei et al., 2016) with an ensemble of word embeddings. [c]Approximate comparison; we did not compare on the same splits. Tang et al. (2015) compares several models and found that a word-level Gated RNN performed best. [d]Approximate comparison. Jaech et al. (2016) and Blodgett et al. (2017) report F1 scores of 91.2 and 92.8, respectively, using an LSTM + CNN model.

middle epoch of the best three-epoch dev-accuracy window.

Using the SemEval 2017 dataset we conducted a grid search over the hyperparameter settings listed above. For each RNN and CNN architecture, we find the best hyperparameter setting based on dev set accuracy. We then perform a second grid search over dropout rates from 0 to 0.4 in increments of 0.1. This resulted in a total of 150 dev-set experiments: 45% considering two recurrent architectures and the remainder split between the three convolutional architectures. We perform one test evaluation on each dataset with the best hyperparameters of each architecture.

4 Results

Table 2 shows our results for each model architecture on each dataset. For the RNN baseline, we include the GRU results, which outperformed the LSTM in every experiment. Even though we considered more hyperparameter settings for the RNN models than for any of the CNN architectures, and despite allowing for larger RNN models, each convolutional architecture significantly[4] outperformed the recurrent model. This supports the argument that CNN models are more naturally suited than RNNs to handle the lengthy sequences found in character datasets.

Our models do not achieve state-of-the-art results, in part because we restrict ourselves to character-level models and did not use any of the domain-specific approaches common in evaluations such as SemEval (Rosenthal et al., 2017). The past results we include in Table 2 all outperform our best sequence models. However, many of those results depend upon domain-specific optimizations which are mostly independent of the underlying sequence model.

The simpler n-gram models outperform our best sequence model on four of the five datasets and are quite close to the best results reported by past work. As character n-grams (especially with n=5) are a close approximation to words, this emphasizes the value of explicit word-level features. Bag-of-words or "bag of character ngrams" models naturally model the presence of a specific word, whereas a character sequence model must learn to pick that word out from its surrounding characters. In our sentiment datasets, it may be that specific words are very indicative of the sequence labels, which could in part explain the good performance of the n-gram models. This suggests that models which combine word and character features together may be particularly well-suited to our domain of informal social media texts (Joulin et al., 2017; Luong and Manning, 2016).

Our work has focused exclusively on the domain of social media character sequences, but the need to learn long-distance dependencies is not exclusive to this domain. Modeling large documents, while traditionally done at the word level, can involve very long sequences. The tradeoffs we explore between CNNs and RNNs may reappear in such domains.

The two LID datasets demonstrate a clear trade-off between sequence length and vocabulary

[4] Using a two-proportion z-test, the worst CNN model is better than the RNN with $p < .0001$.

size. When considering the data as raw bytes, the sequences are nearly twice as long but have a much smaller vocabulary size. While the CNN models easily handle these byte sequences, the RNN model performs terribly. This may be because the convolutional filters are able to more easily group nearby bytes to learn the character mappings those bytes represent, or simply due to the increased length.

A further practical benefit of the convolutional models is their speed. Across all test datasets, an average training epoch for the CNN-A architecture was 10-20 times faster than that of the RNN architecture. On the SST dataset with a sequence length of 160, this difference was roughly 15 seconds per epoch; on the Yelp dataset with a sequence length of 768, this difference was nearly 30 minutes per epoch.

5 Limitations and Future Work

We have presented an empirical comparison of RNN and CNN model architectures for character sequence classification. Our results indicate that in this domain, convolutional models perform better and train faster. Our experimental results do not, however, give us much insight into *which aspects* of CNN models contribute to higher classification accuracy. Further experiments could help quantify the benefit of dilated convolutions or of wide filter widths, and understand the variations in performance between our CNN architectures.

Our empirical comparison is also limited in focus to informal social media texts. We did not consider any sequences that are either very short (a few words) or are very long (entire documents). We don't know whether our decision to focus on sequences with lengths between 144 and 768 tokens is partially responsible for the trends we report. Aside from our LID experiments, we only consider English-language data with limited character sets and explicit word segmentation. Additional experiments could also explore whether the performance gap between RNNs and CNNs persists in larger datasets with millions of examples.

Acknowledgments

This work was in part supported by the National Institute of General Medical Sciences under grant number 5R01GM114771.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*.

Jeremy Barnes, Roman Klinger, and Sabine Schulte im Walde. 2017. Assessing state-of-the-art sentiment models on state-of-the-art sentiment datasets. In *WASSA*. pages 2–12.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5(2):157–166.

Shane Bergsma, Paul McNamee, Mossaab Bagdouri, Clayton Fink, and Theresa Wilson. 2012. Language identification for creating language-specific twitter collections. In *Proceedings of the second workshop on language in social media*. Association for Computational Linguistics, pages 65–74.

Su Lin Blodgett, Johnny Wei, and Brendan O'Connor. 2017. A dataset and classifier for recognizing social media english. In *W-NUT*. pages 56–61.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* .

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* .

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

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

Aaron Jaech, George Mulcaire, Shobhit Hathi, Mari Ostendorf, and Noah A Smith. 2016. Hierarchical character-word models for language identification. *arXiv preprint arXiv:1608.03030* .

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *EACL*. volume 2, pages 427–431.

Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410* .

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. 2016. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099* .

Nal Kalchbrenner, Aäron Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. 2017. Video pixel networks. In *International Conference on Machine Learning*. pages 1771–1779.

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

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *AAAI*. pages 2741–2749.

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

Tao Lei, Hrishikesh Joshi, Regina Barzilay, Tommi Jaakkola, Kateryna Tymoshenko, Alessandro Moschitti, and Lluís Màrquez. 2016. Semi-supervised question retrieval with gated convolutions. In *NAACL*. pages 1279–1289.

Minh-Thang Luong and Christopher D Manning. 2016. Achieving open vocabulary neural machine translation with hybrid word-character models. In *ACL*. volume 1, pages 1054–1063.

Minh-Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba. 2014. Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206* .

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in python. *Journal of machine learning research* 12(Oct):2825–2830.

Sara Rosenthal, Noura Farra, and Preslav Nakov. 2017. Semeval-2017 task 4: Sentiment analysis in twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. pages 502–518.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew 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.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. Lstm neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075* .

Duyu Tang, Bing Qin, and Ting Liu. 2015. Document modeling with gated recurrent neural network for sentiment classification. In *EMNLP*. pages 1422–1432.

Soroush Vosoughi, Prashanth Vijayaraghavan, and Deb Roy. 2016. Tweet2vec: Learning tweet embeddings using character-level cnn-lstm encoder-decoder. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, pages 1041–1044.

Yichun Yin, Yangqiu Song, and Ming Zhang. 2017. Nnembs at semeval-2017 task 4: Neural twitter sentiment classification: a simple ensemble method with different embeddings. In *SemEval*. pages 621–625.

Fisher Yu and Vladlen Koltun. 2015. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122* .

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*. pages 649–657.