An Evaluation of Neural Network Models for Toxic Comment Classification

Lucas Sterckx
lucas.sterckx@gmail.com

Abstract

Today, online discussion has become an integral part of people's interactions, e.g., through commenting interfaces as well as the various popular social media platforms. Alas, these channels also suffer from abuse in the form of online harassment through so-called toxic comments. In this paper we provide a systematic evaluation of neural network based text classifiers for the task of automatically detecting such toxic comments and classifying them according to the type of toxicity. We present a large-scale analysis of the sensitivity of the prevalent recurrent and convolutional neural network architectures to hyperparameters. We report a comprehensive experimental evaluation of those state-of-the-art models on the Kaggle Toxic Comment Classification Challenge dataset. Our experiments provide practical insights into the relative importance of factors such as neural network type, pretrained embeddings, network depth, RNN cell type, attention, regularization and sensitivity to initialization. We find that baseline convolutional and recurrent neural networks are competitive in terms of evaluation metrics. However, convolutional neural networks show less sensitivity to initialization and hyperparameter selection. A crucial aspect of neural network based classifiers is the use of pretrained embeddings which are most effective when trained on web-scale corpora.

1 Introduction

With the rise of social media platforms, online communication and discussion has become an essential part of people's internet experience. Unfortunately, online discussion is also an avenue for abuse. A 2014 Pew Report highlights that 73% of adult internet users have seen someone harassed online, and 40% have personally experienced it (Duggan, 2014). The threat of abuse, bullying and harassment online means that people stop expressing themselves and give up on seeking different opinions. Toxic comments are comments that are rude, disrespectful or otherwise likely to make someone leave a discussion.

Currently, online platforms struggle to effectively monitor conversations for toxic behavior, to the extent that many communities limit or completely shut down user comments. Therefore, fast identification of toxic comments and prediction in real time is of paramount importance, to prevent detrimental effects of such toxic behavior on well-intentioned internet users.

Recently, the Conversation AI team[1], a research initiative founded by Jigsaw and Google (both part of Alphabet), is working on tools to help improve online conversation. An area of focus for Conversation AI is the study of negative online behaviors, such as toxic comments. So far they have built a range of publicly available models for monitoring text served through the *Perspective* API (Hosseini et al., 2017). These models are still error-prone and do not allow users to select which types of toxicity they are interested in classifying (e.g., some platforms may be fine with profanity, but not with other types of severe toxic content).

Such text classification, in its general form, is a classic topic for natural language processing and an essential component in many applications, such as web search, information filtering, topic categorization and sentiment analysis. As a result, a wide range of machine learning methodologies have been applied for text classification. Recent research in the are has mostly focused on the application of neural network (NN) architectures. The abundance of NN architectures for text classification includes two dominant groups: convolutional neural networks (CNNs) and recurrent neural net-

[1] https://conversationai.github.io

works (RNNs). Keeping track of literature in this domain is challenging, and an overview of the many architectural variants for text classification, each with many of their specific hyperparameters, is largely missing from literature. We try to fill this gap, an thus hope to enable overcoming the expensive and slow development of such classifiers for toxic comment identification. Indeed, for developers of new systems, sweeping the large hyperparameter spaces — even though common in computer vision (Huang et al., 2017) — would be prohibitively expensive. In practice, researchers often limit themselves to well-established architecture and hyperparameter choices.

We will present an extensive experimental comparison of RNN and CNN based architectures and their hyperparameters. Thus, our work has a similar objective as work by Zhang et al. (2017) for convolutional neural networks for text classification and Britz (2017) for neural machine translation. We will use a recently released dataset from the Kaggle Toxic Comment Classification Challenge[2], described in the next Section 2. Further extending the landscape of model architectures is beyond the scope of this work: we rather provide an analysis of the existing ones as they also show to outperform more complex ones on many occasions. An overview of the prevalent CNN and RNN architectures is presented in Section 3. Our main contribution is a systematic comparison of those architectures, presented in Section 4.

From our comparison we conclude that RNN and CNN based classifiers are competitive for the task. However, we find that RNNs are more sensitive to overfitting initialization. Pretrained embeddings are shown to be a crucial for NN based classifiers. Variants which are trained on web-scale corpora of text are evaluated as most effective. Regularization via data augmentation can provide a boost in performance by reducing the models' capacity to overfit on the training data. We make all our code available online.[3]

2 Toxic Comment Classification Challenge

Unfortunately, abuse and harassment online has been shown occur increasingly frequently (Duggan, 2014), especially with the rise of social media platforms and the explosion of online communication. This has a detrimental effect on the activity of well-intentioned users: indicatively, the Wikimedia foundation found that 54% of those who had experienced online harassment expressed decreased participation (Wulczyn et al., 2017). Moderation to identify and eventually block such toxic comments would thus be most welcome, but proves to be hard to effectively implement. Despite efforts to enhance the safety of online environments based on crowdsourcing voting schemes or the capacity to denounce a comment, in most cases these techniques are inefficient and fail to predict a potential toxicity (Hosseini et al., 2017). A main limitation of current models such as the Perspective API is that they are not yet sufficiently reliable (i.e., precision or recall is too low), and that usually the degree of toxicity is not determined (i.e., binary classification is often too coarse for practical purposes).

Thus, to stimulate research on toxic comment identification, Wulczyn et al. (Wulczyn et al., 2017) released a dataset of discussion comments from English Wikipedia and proposed a methodology that combines crowdsourcing and machine learning to analyze personal attacks at scale.

To stimulate research on more precise and versatile detection, the Toxic Comment Classification Challenge was organized by Google and Conversation AI, managed through Kaggle. A crowdsourced dataset based on (Wulczyn et al., 2017) that includes 6 toxicity sub-types (reasons why something might be considered toxic), and approximately 160k human labelled comments from Wikipedia Talk pages. The labelled annotations are based on asking crowd-workers to rate Wikipedia comments according to their toxicity (likely to make others leave the conversation). This dataset provides the most reliable evaluation of toxic comment classification to date, and is sufficiently large to apply powerful models such as NN architectures.

In the challenge, participants need to build a multi-label classification model that is capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based hate, providing probability estimates for each sub-type, ideally strong enough to outperform Perspective's current models. The dataset comprises comments from Wikipedia's talk page edits, which we next analyze descriptively.

[2] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge

[3] http://www.github.com/anonymized

	Training / Test
Vocabulary size	220,340
# Comments	95,851/63,948
# Tags	21,195/14,015
⊘ Tokens per comment	80/79

Table 1: Properties of Toxic Comment Classification dataset (⊘ denotes the average amount).

2.1 Dataset

In this section we provide a brief exploratory data analysis of the dataset and illustrate some key properties of the text collection. We facilitate our exploratory data analysis with the excellent scripts by participants of the toxic comment classification challenge[4].

During the competition leakage of the initial test collection was detected because of overlap with one of the existing datasets released by Conversation AI. A month into the challenge a new test collection was released and the submission deadline was extended. The label distribution of the new test collection diverted strongly from that of the training collection and the initial test collection due differences in annotation style, resulting in an unreliable development of new classifiers and poor application of existing ones. In our experiments, we use the initial dataset, released at the start of the Kaggle competition, which is smaller than the final one, but offers more reliable development of the classifiers.

Figure 1 shows the amount of labels for each type of toxicity assigned to the comments. On average 20% of the comments in the training data receive at least one of the six tags indicating toxicity, whereby a subset of 20 comments are marked with all of the categories. Table 1 shows properties of the dataset such as the amount of documents and the vocabulary size. Figure 2 shows the distribution of the amount of tokens for varying lengths of comments, with an average of 80 tokens per comment.

3 Model architectures

In this section we present two prevalent NN architectures: convolutional and recurrent neural networks.

[4] https://www.kaggle.com/jagangupta/stop-the-s-toxic-comments-eda

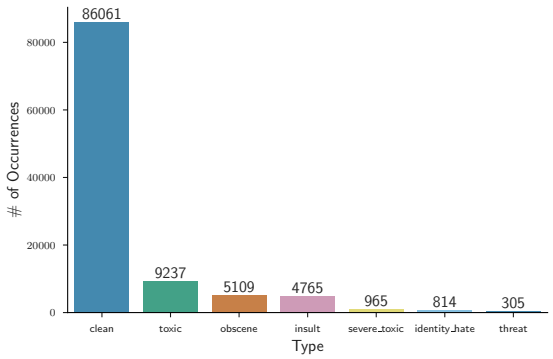

Figure 1: Distribution of the labels.

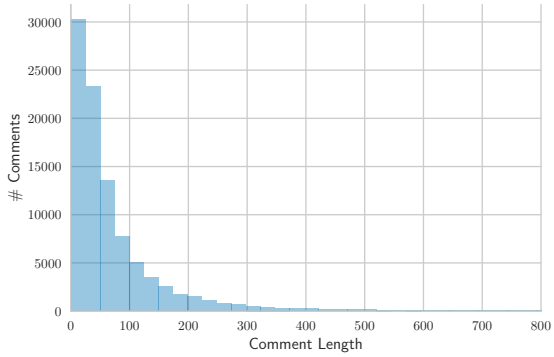

Figure 2: Distribution of comment length.

3.1 CNN Architectures

Convolutional neural networks (CNNs), originally proposed by Lecun et al. (1998) are neural networks that model the internal structure of data such as the 2-dimensional structure of image data through convolutional layers, where each computation unit responds to a small region of input data (e.g., a small square of a large image). The essence of CNNs is to convert small regions of data into feature vectors for use in the upper layers: a convolutional layer learns to transform small regions of the data into higher level feature vectors.

The architecture proposed by Kim et al. (Kim, 2014) has become one the most applied methods to apply CNNs to text and has shown to be effective on many occasions, achieving state-of-the art results in sentiment classification. As a baseline system, we apply a similar architecture to the one described by Kim et al. (2014). Our model consists of a single convolutional layer with multiple filter sizes, followed by one fully connected layer and a sigmoid over the six-dimensional binary label vector.

3.2 RNN Architectures

Recurrent neural networks (RNNs) (Elman, 1998) process sequences of arbitrary length by recursively applying a nonlinear transition function to an internal hidden state vector h_t and the current input item x_t.

3.2.1 RNN Types

LSTM The long short-term memory network (LSTM) was first proposed by Hochreiter et al. (1997) to specifically address this issue of learning long-term dependencies in vanilla RNNs. The LSTM maintains a separate memory cell inside it that updates and exposes its content only when necessary. h_t is a "candidate" hidden state that is computed based on the current input and the previous hidden state h_{t-1}. The LSTM contains three gates: the input, forget and output gates. c_t is the internal memory or context vector of the unit. It is a combination of the previous memory c_{t-1} multiplied by the forget gate, and the newly computed hidden state h, multiplied by the input gate.

GRU A gated recurrent unit (GRU) (Cho et al., 2014) has but two gates, a reset gate and an update gate, and thus has a smaller amount of parameters. The reset gate determines how to combine the new input with the previous memory, and the update gate defines how much of the previous memory to retain. GRUs do not possess internal context states (c_t) that differ from the exposed hidden state, and do not have the output gate that is present in LSTMs. The input and forget gates are coupled by an update gate and the reset gate is applied directly to the previous hidden state. Thus, in a GRU, the LSTM reset gate functionality is really split up into both the reset and update gate.

BiRNN Bidirectional RNNs encode the considered sequence (i.e., the comment) after processing it in a forward and backward direction by a different RNN, and concatenating the separate hidden states.

3.2.2 Pooling

After a forward (and backward) pass through the network, RNNs provide us with a sequence of contextual embeddings of the tokens in the comment. To further reduce the dimensionality of this representation, a *pooling* operation is performed. We experiment with four different ways of merging the contextual embeddings to a low-dimensional comment representation. We add a single fully connected layer on top of this representation from the RNN.

Max-Pooling vs. Average-pooling The most commonly applied pooling operation for RNNs (and CNNs) is the max-pooling, which selects the maximum value across each hidden dimension in the sequence of RNN (or CNN) outputs. The average-pooling operation instead calculates the averages over the outputs.

Self-Attention Attention is used in sequence-to-sequence models to attend over states of an encoder RNN, but can also be used in any sequence model to look back at past states. The attention function $f_{att}(h_i, s_j)$ calculates an unnormalized alignment score between the current hidden state h_i and the previous hidden state s_j. Without any additional information, however, we can still extract relevant aspects from the sentence by allowing it to attend to itself using self-attention (Lin et al., 2017). Self-attention, also called intra-attention has been used successfully in a variety of tasks including reading comprehension (Cheng et al., 2016), textual entailment (Parikh et al., 2016), and abstractive summarization (Paulus et al., 2017).

CNN As a final method to generate single representation from the RNN states, we use another parameterized network. We experiment with combining the RNN and CNN by placing the CNN architecture, discussed in Section 3.1, on top of the contextual word representation generated by the RNN instead of the word embeddings. We place the CNN-encoder on top of the hidden states $h_1, , h_m$ of the RNN, and jointly train the whole architecture.

3.3 Pretrained Word Embeddings

The first layer of NN architectures embeds the one-hot token representations into a vector space of lower dimensionality, which it then fine-tunes through back-propagation. This layer yields *word embeddings* as the weights of the first layer and is usually referred to as the *embedding layer*. Word embeddings learned without supervision have seen tremendous success in numerous NLP tasks in recent years. Pre-training of the embedding layer using a dedicated word embedding technique such as Word2Vec (Mikolov et al., 2013) or Glove (Pennington et al., 2014) has proven to be an effective method to distill knowledge from large corpora. We collect a set of

pretrained word embeddings from various online sources and use them to initialize the word embedding layer of the classifiers. Next to the training procedure these pretrained embeddings differ in source of text, dimension, amount of tokens

Word2vec[5] Word2Vec by Mikolov et al. (2013) is arguably the most popular of the word embedding models. The CBOW model learns word representations by predicting a word according to its context. The context is defined as a symmetric window containing all the surrounding words. The Skip-Gram model, the most commonly applied version of Word2Vec, uses the centre word to predict the surrounding words to the left and to the right of the target word.

We use 300 dimensional Skip-Gram Word2Vec embeddings pre-trained on the Google News corpus (3 billion tokens).

Dependency Based Word2Vec[6] Omer and Levy (2014) generalize the Skip-Gram model, and move from linear bag-of-words contexts to arbitrary word contexts. Dependency-based word embeddings use syntactic contexts derived from automatically produced dependency parse-trees.

We use the dependency based word embeddings released by the authors trained on English Wikipedia with 300 dimensions.

GloVe[7] Pennington et al. (2014) illustrate that the ratio of the co-occurrence probabilities of two words (rather than their co-occurrence probabilities themselves) is what contains information and so look to encode this information as vector differences. For this to be accomplished, they propose a weighted least squares objective that directly aims to reduce the difference between the dot product of the embeddings of two words and the logarithm of their number of co-occurrences. We use GloVe embeddings trained on three different sources of text: Twitter text (200 dimensions), WikiNews (300 dimensions) and general web text from the Common Crawl corpus (300 dimensions).

fastText[8] Mikolov et al. (2018) train word vector representations by using a combination of tweaks that were previously not used together. FastText uses the standard Word2Vec CBOW model as basis and applies position-dependent weighting, enriches the model with word n-grams and a bag-of-character n-gram vectors.

We use fastText embeddings trained on two different sources of text: WikiNews (300 dimensions) and general web text from the Common Crawl corpus (300 dimensions).

3.4 Regularization

Because NN architectures are heavily parameterized, they are prone to overfitting. An important aspect of classifier development to prevent this from happening is regularization. We apply two established methods for regularization of the network weights: dropout and data augmentation.

Dropout Dropout (Srivastava et al., 2014) is the most applied form of regularization for NN architectures. During training, a subset of network nodes are removed with a probability p. At each train iteration, only such a reduced network is considered. When applying the network for predictions, no nodes are dropped, and all learned weights contribute to the outputs.

Data Augmentation An alternative way to prevent overfitting is data augmentation. We can increase generalization of machine learning models by simply training on more data. A way to get around the problem that more labeled data is often not available, is to artificially generate extra data based on the available data, and add it to the training set. We evaluate a method for data augmentation proposed by one of the participants of the challenge.[9] The method is similar to the back-translation proposed by Sennrich et al. (2016). The English comments are translated to an intermediate language using a pre-trained translation model, and afterwards translated back to English. This way, we extend the original training with paraphrases. We use three different intermediate languages: German, Spanish and French. We use the Google Translate API as translation model. During training, we mix synthetic training data into the original and do not distinguish between the two.

[5] https://code.google.com/archive/p/word2vec/

[6] https://levyomer.wordpress.com/2014/04/25/dependency-based-word-embeddings/

[7] https://nlp.stanford.edu/projects/glove

[8] https://ronan.collobert.com/senna/

[9] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/discussion/48038

4 Experiments

4.1 Experimental Setup

We develop classifiers on a development set which contains 10% from the original training data. We restrict the vocabulary to 120,000 tokens and threshold the maximum length of the comments to 500 tokens.

For CNN-based architectures we allow weights of the embedding layer to be adapted during training, for RNN-based architectures we leave them fixed. All RNN and CNN models were implemented using Pytorch.[10] We use 32 dimensional embedding as hidden layer between the RNN and CNN representations of the comments.

We train all models using Adam (Kingma and Ba, 2014) with an initial learning rate of 0.001, decayed after 2 epochs to 0.005, these rates are tuned on the development set. We train models for 4 (CNN) and 3 (RNN) epochs without early stopping, overall the model were shown to overfit very fast. We use mini batches of 32 training instances.

Baselines As non-neural baseline we use a logistic regression classifier trained on concatenated token n-grams (unigrams, bigrams and trigrams) and character n-grams (3 to 6 characters) representations of the comments, both weighted using their corresponding inverse document frequency values. Next to one-hot representations we include several features proposed by other participants in the challenge such the amount of punctuation, the amount of capital letters, unique words, and others. Further details are not deemed relevant, as the inclusion of these features does not affect performance significantly. We do observe a large boost for all evaluation metrics by inclusion of character n-gram features. Table 2 shows scores for our baseline model.

	AUC	LL	Acc.
LR + Bag-of-Words	97.845	0.0543	98.111
+ Character N-grams	98.502	0.0472	98.280
+ Feature Design	**98.503**	**0.0472**	**98.282**

Table 2: Logistic regression baseline model

4.2 Evaluation

We report on 3 evaluation metrics: the area-under-the-curve (AUC), log-loss (LL) and accuracy. The final evaluation metric used in the toxic comment challenge was the AUC, as it is less sensitive to the difference in probability distribution between training and test data than the log-loss, originally proposed as evaluation metric. The AUC is less sensitive to absolute values of the probability estimates and instead emphasizes the ranking of the probabilities relative to one another.

[10] http://pytorch.org/

Filter Sizes	# Filters	AUC	LL	Acc.
3	64	97.810	0.0473	98.198
3	128	98.202	0.0450	98.314
3	256	98.573	0.0435	98.347
3	512	98.640	0.0431	98.347
3,4,5	64	98.353	0.0440	98.322
3,4,5	128	98.672	0.0432	98.348
3,4,5	256	98.747	**0.0423**	98.371
3,4,5	512	**98.799**	0.0431	**98.376**
3,4,5,6,7	64	98.616	0.0430	98.338
3,4,5,6,7	128	98.753	0.0429	98.359
3,4,5,6,7	256	98.796	0.0429	98.355
3,4,5,6,7	512	98.788	0.0442	98.368

Table 3: CNN hyperparameter selection

Type	Layers	Dim.	AUC	LL	Acc.
BiGRU	1	32	98.120	0.0449	98.325
BiGRU	1	64	**98.741**	0.0426	**98.363**
BiGRU	1	128	98.721	0.0426	98.360
BiGRU	1	256	98.685	0.0435	98.316
BiGRU	2	32	98.383	0.0430	98.355
BiGRU	2	64	98.382	0.0431	98.343
BiGRU	2	128	98.467	0.0428	98.335
BiGRU	2	256	98.405	0.0437	98.346
BiLSTM	1	32	98.097	0.0457	98.271
BiLSTM	1	64	98.591	0.0429	98.362
BiLSTM	1	128	98.536	**0.0419**	98.390
BiLSTM	1	256	98.405	0.0437	98.346
BiLSTM	2	32	98.210	0.0442	98.296
BiLSTM	2	64	98.365	0.0429	98.343
BiLSTM	2	128	98.317	0.0431	98.350
BiLSTM	2	256	98.045	0.0447	98.320

Table 4: RNN hyperparameter selection with Max-pooling and Glove 300-dimensional embeddings

5 Discussion

In this section we discuss the influence of the parameters presented in previous sections of neural network architectures for classification.

5.1 Performance of CNNs

Results for different configurations of CNNs are shown in Table 3. We consider region sizes of 3, 4, 5, 6, and 7, and vary the amount of filter maps per size as 64, 128, 256, and 512. The CNN architectures benefit from larger amounts of filters for

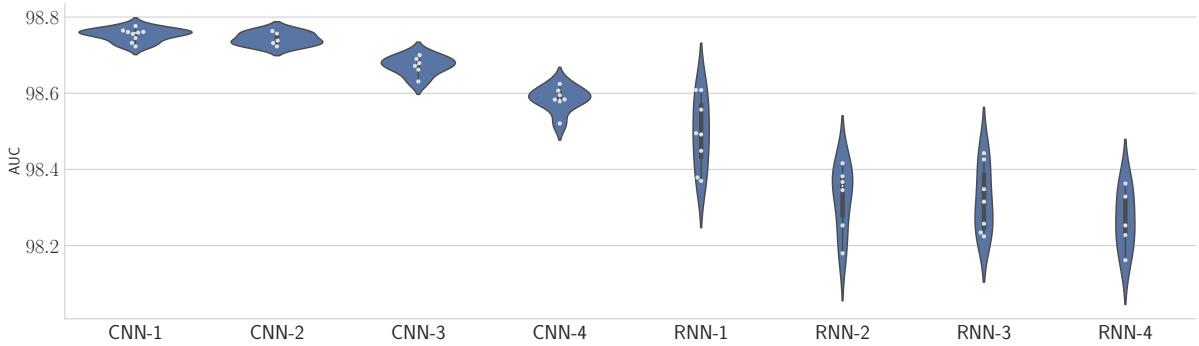

Figure 3: Distribution of AUC scores when re-training the four highest scoring RNN and CNN configurations multiple times with different seed values.

	AUC	LL	Acc.
CNN (128 filters)	97.100	0.0505	98.187
+ Word2vec	98.585	0.0447	98.317
+ Dep. Word2Vec	98.056	0.0469	98.274
+ Glove Wikinews	98.580	0.0438	98.339
+ Glove Twitter	98.635	0.0435	98.342
+ Glove CommonCrawl	**98.766**	0.0427	98.361
+ fastText Wikinews	98.569	0.0438	98.347
+ fastText CommonCrawl	98.748	**0.0423**	**98.362**
BiGRU (2 layers, 128)	97.548	0.0556	98.166
+ Word2Vec	98.389	0.0461	98.207
+ Dep. Word2Vec	97.901	0.0559	98.312
+ Glove Wikinews	98.387	0.0460	98.243
+ Glove Twitter	98.549	0.0485	98.313
+ Glove CommonCrawl	**98.636**	0.0429	**98.366**
+ fastText Wikinews	98.388	0.0460	98.189
+ fastText CommonCrawl	98.595	**0.0425**	98.340

Table 5: Pretrained embeddings for CNN (128 filters, sizes 3,4,5,6,7) and BiGRU (hidden dim. of 64) hyperparameter selection.

	AUC	LL	Acc.
Self-Attention	98.242	0.0460	98.282
Average	98.459	0.0450	98.327
CNN	98.649	0.0427	98.332
Final State	98.691	0.0432	98.361
Max-pooling	**98.741**	**0.0426**	**98.363**

Table 6: Pooling-layers for a single-layer BiGRU architecture with a hidden dimension size of 64.

	AUC	LL	Acc.
BiGRU (1-layer 128-dim)	98.657	0.0440	98.390
+ Dropout	98.721	0.0426	**98.360**
+ Data Augmentation	**98.791**	**0.0421**	98.354
CNN	98.740	0.0447	98.302
+ Dropout	98.799	**0.0430**	**98.376**
+ Data Augmentation	**98.828**	0.0432	98.305

Table 7: Effect of data-augmentation.

more different sizes.

5.2 Performance of RNNs

Results for different configurations of RNNs are shown in Table 4. Overall, GRUs obtain higher scores than LSTMs. Little improvement is obtained by more layers and larger hidden states (> 64). While some RNN configurations match CNNs in terms of evaluation metrics, we found training more brittle and sensitive to overfitting.

5.3 Pooling

Results for different pooling methods for RNNs are shown in Table 6. Surprisingly, regular max-pooling and the final hidden state stay one of the most effective pooling strategies. We were unable to achieve higher scores using parameterized pooling layers on top of the hidden states such as self-attention and CNNs.

5.4 Pretrained Embeddings

Results for all embeddings discussed in Section 3.3 are shown in Table 4. We found word embeddings to be crucial for NN architectures to have scores competitive with the baseline linear classifiers. The Glove and fastText embeddings trained on very large corpora (> 6 billion tokens) are found to be the most suited embeddings for toxic comment classication.

5.5 Regularization

Results for dropout an data augmentation are shown in Table 7. For RNNs we use dropout on the embedding layer and in between the layers of multi-layer RNNs. For CNNs we apply dropout only on the final fully connected layer. Our standard models apply a dropout mask with probability of $p = 0.1$ on the connections between the RNN layers and on the fully connected layer at the

output. Both dropout and the data augmentation strategy improve the AUC scores considerably.

5.6 Initialization

Reimers et al. (2017) show that reporting a single performance score is insufficient to compare the non-deterministic approaches that are NNs. They demonstrate that the seed for initialization has a statistically significant impact on the test performance and that wrong conclusions can be made if performance scores based on single runs are compared for two named entity recognition tasks.

To study the sensitivity of the models to initialization, we also retrain models multiple times with different seed values. Similarly to the findings described above, we observe high variance of classification scores depending on initialization using different seeds. Figure 3 shows violin plots similar to Reimers et al. (2017). For both CNNs and RNNs we show the four best performing configurations. From the violin plots it is apparent that RNN architectures show much greater sensitivity to different initialization seeds than CNNs.

6 Winning Entry

At the time of writing the Toxic Comment Classification Challenge has ended and authors of the top-submission have kindly shared their methodology.[11] We briefly describe their approach and compare it to ours.

Their findings are similar to ours in that simple models such as the ones presented here, perform well for toxic comment classification. Ensembling of models was shown to be very effective in the competition and essential to produce top-ranking submissions.

While we highlight the effectiveness of CNNs, the ensemble of the winning submission consists primarily of BiGRUs, with only a couple of CNNs included. Reasons for this could be the second fully connected layer on top of the pooling layer and the increase in training data after the leakage. No hyperparameter search is reported for CNNs. The developers stress the use of diverse, high dimensional pre-trained embeddings to build effective ensembles as well. The data augmentation strategy discussed in Section 3.4 was included in the winning submission and reported to increase AUC scores of the BiGRU models with 0.005.

[11] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/discussion/52557

7 Conclusion

Both the industrial and research community have tried to efficiently identify online toxic comments, given their important (negative) effects on in online interactive user communications. We focused on recent neural network (NN) architectures for toxic comment classification and presented a performance comparison. In particular, we focus on CNN and RNN architectures with a selection of different pretrained word embeddings, as well as regularization methods. We study the effect of network width, depth, pooling operations, embeddings and regularization. We summarize our findings as follows:

- Simple NN architectures outperformed feature-based methods.
- Pretrained embeddings were essential for optimal results with NN architectures. Pretrained Glove and fastText embeddings trained on webscale text collections (6B+ tokens) were shown to outperform other variants.
- Shallow NN architectures outperformed deeper ones. Depth added beyond 2 RNN layers quickly deteriorates performance.
- Overall, RNN architectures performed slightly worse than CNN architectures. However, CNN architectures were less prone to hyperparameter sensitivity and thus offers lower risk for overfitting.
- Added complexity in pooling layer did not improve performance.
- Regularization such as dropout and data augmentation were shown to be beneficial.

We hope that these guidelines help to build more effective toxic classification systems in the future, at faster development time.

References

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1442–1451.

Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*.

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*.

Maeve Duggan. 2014. *Online harassment*. Pew Research Center.

Jeffrey L Elman. 1998. *Rethinking innateness: A connectionist perspective on development*, volume 10. MIT press.

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

Hossein Hosseini, Sreeram Kannan, Baosen Zhang, and Radha Poovendran. 2017. Deceiving google's perspective api built for detecting toxic comments. *arXiv preprint arXiv:1702.08138*.

Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. 2017. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751.

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

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Omer Levy and Yoav Goldberg. 2014. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 302–308.

Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*.

Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

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.

Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*.

Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Nils Reimers and Iryna Gurevych. 2017. Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 338–348.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 86–96.

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.

Ellery Wulczyn, Nithum Thain, and Lucas Dixon. 2017. Ex machina: Personal attacks seen at scale. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 1391–1399, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

Ye Zhang and Byron Wallace. 2017. A sensitivity analysis of (and practitioners guide to) convolutional neural networks for sentence classification. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 253–263.