Text Classification performance boosting by improved Easy Data Augmentation

Xusheng Ji, Meiying Chen
Goergen Institute for Data Science
University of Rochester
$\{xji6, mchen71\}$ @$ur.rochester.edu$

Abstract

In this paper, we exploit the benefits of data augmentation working on a text classification problem. We improved an existing data augmentation algorithm which called easy data augmentation (EDA) to allow it to generate number of augmentation and percent of changed words automatically. Additionally, we reduce the risk of sentence meaning changed by doing operation dropping with some probabilities depending on the length of the sentences. Furthermore, we experimentally evaluate the performance gain by running three advanced deep learning models TextCNN, Bidirectional LSTM and Bert Transformer on Kaggle Toxic Comments Classification Challenge. We found that the improved version of EDA can boost performance on each of the three deep leaning models.

Introduction

Text classification is an important part of the natural language processing research and it is invariably used in several research area such as natural language understanding in dialogue system (Grau et al.), sentiment analysis(Wilson, Wiebe, and Hoffmann) and scammers identifying(Stabek, Brown, and Watters 2009). However, the performance of text classification tasks is always inextricably linked with the data size and data quality. Data augmentation can be treated as an excellent method to improve performance in computer vision tasks(Bérard 2003) and some researchers also apply it to the text classification tasks, such as translate back(Zhang and LeCun 2016), synonym insertion(Topkara, Topkara, and Atallah) and GAN(Frid-Adar et al. 2018). However, since all the methods above are costly and time-consuming, it is essential to discover a new method in the text classification task.

One famous study developed a new algorithm called Easy Data augmentation(EDA)(Wei and Zou 2019) to augment data by inserting synonym, replacing synonym, swapping words and deleting words(Wei and Zou 2019). But this method can work little if anything on large datasets and the number of long sentences is not considered. In this paper, we improved the EDA by considering a small number of long sentences and also reducing the probability of meaning changed.

Another approach to alleviating the negative influence of data scarcity and low-quality issues is to introducing pre-training procedures. Several pre-trained language representation models have been put forward in the past few years. BERT(Bidirectional Encoder Representations from Transformers) is a good example(Devlin et al. 2019). In this project, we modified BERT to adapt our aimed task.

We evaluated our proposed approaches on a classical text classification task which is the Kaggle toxic comments challenge(van Aken et al. 2018). Three updated models: TextCNN, bidirectional Long Short-Term Memory Networks(LSTM) and Bert Transformer will be discussed. The results show that our updated version can work well on this dataset. Our code will be publicly available.

Related Work

Data augmentation has been a vital research area in data mining and machine learning. In the area of computer vision, some efficient data augmentation techniques such as flipping and rotation can always improve the performance dramatically with deep learning models. Because they always require a huge amount of data to train themselves (Taylor and Nitschke 2017). However, only a few studies have been done in the area of data augmentation with NLP tasks since most of the techniques for it demand a high cost for implementation. In this part, we examine some prominent techniques for data augmentation and text classification on NLP tasks.

Back Translation

Sergey and Myle(Edunov et al. 2018) managed to combine beam search and unrestricted sampling to augment the training language sentences. They discovered that neural machine translation can be improved by adding noisier synthetic sentences than genuine bitext when translating them back. They evaluated the performance with BLEU score and it showed that beam with noise data can work best among greedy search and beam search when conduct back translation with more than 17 million sentences. However, most of the translation tools will not be free and invariably have words limitation. Furthermore, even some translation interfaces can only support limited languages such as English and Chinese. As a consequence, back translation can hardly be utilized in practice.

DAGAN

Antreas and Amos(Antoniou, Storkey, and Edwards 2018) trained a form of data augmentation generative adversarial network(DAGAN) on the vanilla classifier to generate a huge amount of example images. The idea is that the effect of true and false can be achieved by iteration and enhancing the training phase in the way of contesting between generator and discriminator. However, the difficulty of this method is that high-quality data can be achieved only when the GAN model is well trained, which always leads to relatively complicated problems and demanding an ocean of work.

EDA

Synonym insertion has been an old and useful technique for data augmentation in NLP tasks. In 2019, Jason and Kai(Wei and Zou 2019) from Dartmouth College proposed another three mechanisms which are synonym replacement, random swap, and random deletion combine with synonym insertion to generate the EDA algorithm. They performed all these four methods on five different dataset including Stanford sentiment treebank(Socher et al.), customer reviews(Hu and Liu), question type dataset(Li and Roth 2006), subjectivity/objectivity dataset(Pang and Lee 2004) and Pro-Con dataset(Jindal). Then they tried two excellent deep learning modes which are RNN and CNN on four groups of experimental training set with various sample sizes ranging from 500 to 10000. Lastly, they revealed that average accuracy improved by about 5% on the small training dataset with 500 samples and less than 1% on the full training dataset. EDA can help avoid overfitting and boost the performance, but in their research, we realized that they did not focus on the fewer longer sentences in some special cases and the method can work little but anything on a large dataset with only around 10000 samples.

Methodology

In this project, we modified the source code of EDA so that it can generate the number of augmentation by the length of sentences instead of passing the augmentation parameter manually. Then the short sentences will have a higher probability for dropping each of the three operations including synonym insertion, random swap, and random deletion to reduce the risk for meaning changed.

Correspondingly, we downloaded the dataset from Kaggle toxic comments challenge to be our training set and test set. Finally, we use three state-of-art models which is TextCNN(Xiong et al. 2018), bidirectional Long Short-Term Memory Networks(LSTM)(Cheng, Dong, and Lapata 2016) and Bert Transformer(Devlin et al. 2019) to train the dataset with and without updated EDA. Additionally, we apply glove(Pennington, Socher, and Manning 2014),word2vec(Rong 2016) and fasttext(Liao et al. 2016) to be the word embedding files for TextCNN and LSTM. In this way, we can evaluate the performance of our updated data augmentation algorithm working on this large dataset for solving a text classification problem.

Improved version of Easy Data Augmentation(EDA)

In the original version(Wei and Zou 2019), the number of augmented sentences (Naug) is a parameter passed by users manually and it ranges from 1 to 32 depending on the size of dataset. This mechanism only works on the whole dataset instead of taking number of long sentences and short sentences into consideration. As a consequence, we removed the number of augmented sentences and from the parameter list, where is a parameter which describe the percent of words will be changed within one sentence. Then we generated number of augmentation and depending on the size of dataset automatically since Jason (Wei and Zou 2019) indicated that these two parameters can be inferred from length of sentence (Wei and Zou 2019). Moreover, since only 28199 sentences from our training set have more than 100 words (we have 159571 comments in total) then we set 100 words in each sentence to be a threshold to identify whether the sentence is long or short. If the sentence is a short one, we generated less augmented sentences than N_{aug} and more augmented sentences than N_{aug} when it is a long one.

Another problem of the authentic version (Wei and Zou 2019) is that although most of the augmented sentences retained their original label, there are still some sentences that will change their meaning that will hurt the model performance. Corresponding to the paper (Wei and Zou 2019), each of the operations can work well alone when they have a little percent of changed words but will hurt the performance of models when the percent is large. Consequently, we added a new parameter called the probability of operations dropping (pod) to drop each of the three operations including random insertion, random swap, and random deletion. Because these three operations will result in the risk of the label changed. The same thing as handling the N_{aug} problem we just discussed above, if the comment is a long one then the probability for operations dropping should be low otherwise there will be a high probability for operation dropping.

Baseline Model - 2D TextCNN

Yoon Kim(Kim 2014) proposed a new Convolutional Neural Networks to solve the sentence classification problem and it is called TextCNN. The main idea is to use multiple kernels of different sizes to extract key information in sentences and it is very similar to multiple window N-gram(Abou-Assaleh et al.). In such a manner, the model can capture the local correlation better. Peng and Zhengyu(Zhou et al.) achieved greater performance when combining Bidirectional LSTM(Cheng, Dong, and Lapata 2016) with two-dimensional max-pooling since two dimensional max-pooling can sample more important information when solving sequential problem(Zhou et al.). In this project, we constructed four layers TextCNN as shown in figure 1. Each layer assign with a two-dimensional convolutional layer and a two-dimensional max-pooling layer. And we use Elu(Wang, Qin, and Zhu 2017) to be our activation function in the internal layers because this function can pro-

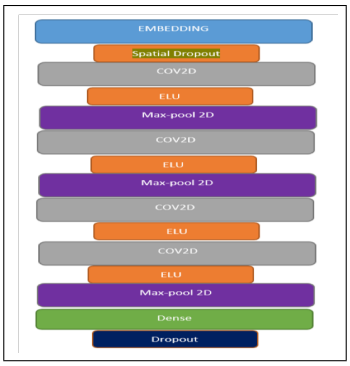

Figure 1: Two-dimensional convolutional layer and two-dimensional max-pooling layer with ELU activation function.

duce more accurate results with a faster converge speed to zero. Figure 1 shows our architecture of TextCNN.

In the convolutional layers, we use 32 filters in total and filter size of 1,2,3,5 for each layer respectively. In terms of the two-dimensional convolutional layers, each layer will assign a 2D filter with n-words and k-feature vectors. Then the feature $F_{i,j}$ will be generated from a filter vector $H_{i:i+n-1,j:j+l-1}$ by following function:

$$F_{i,j} = f(H_{i:i+n-1,j:j+l-1} + b) \tag{1}$$

In the 2D max-pooling layers, we set the vertical pooling size to be the difference between max words length and filter size plus one and horizontal pooling size to be 1. Then if we denote the pooling size to be (P1, P2), then P1 and P2 should be equal to :

$$(P1, P2) = (maxlen - filter\_size + 1, 1) \tag{2}$$

Then max-pooling function can extract the maximum value from the window of matrix F and the pooling results should be shown as follow:

$$P_{i,j} = Max - Pooling(F_{i:i+n-1,j:j+l-1} + b) \tag{3}$$

For the output layer, we use several sigmoid functions for each of the output unit. The classifier takes the hidden state h as the input. For example, if the input text is S then the prediction label should be:

$$P(y|s) = Sigmoid(W * h + b) \tag{4}$$

$$Y = argmax(P(y|s)) \tag{5}$$

$$J(w) = \frac{1}{m}\sum_{i=1}^{m} y_i[logh_w(x_i) + (1 - y_i)log(1 - h_w(x_i))] \tag{6}$$

Bidirectional LSTM

Bidirectional LSTM (Bi-LSTM) composed of forward LSTM and backward LSTM. It is often used to model context information in the area of speech recognition (Graves, Jaitly, and Mohamed 2013), sequence tagging (Huang, Xu, and Yu 2015) and named entity recognition (Chiu and Nichols 2016). Furthermore, Bi-LSTM compensate the disadvantage of plain LSTM when solving encoding problem from back to front information. Additionally, Bi-LSTM works well in fine-grained classification such as strong positive sense, weak positive sense, neutral sense, weak negative sense and strong negative sense (Li and Yuan). The LSTM (Cheng, Dong, and Lapata 2016) equations has been shown as follow:

Input Gates:

$$i_t = \sigma(W_{x_i x_t} + W_{h_i} h_{t-1} + W_{c_i c_{t-1}} + b_i) \tag{7}$$

Forget Gates:

$$f_t = \sigma(W_{x_f x_t} + W_{h_f} h_{t-1} + W_{c_f c_{t-1}} + b_f) \tag{8}$$

Cells:

$$C_t = f_t c_{t-1} + i_t tanh(W_{x_c x_t} + W_{h_c} h_{t-1} + b_c) \tag{9}$$

Output gates:

$$O_t = \sigma(W_{x_o x_t} + W_{h_o} h_{t-1} + W_{c_o c_t} + b_o) \tag{10}$$

Cell Output:

$$h_t = O_t * tanh(C_t) \tag{11}$$

σ is the activation function and it is the Relu (Li and Yuan) function in this case. i, f, c, o represent the input gate, forget gate , cell and output gates respectively. In this project, we use 50 LSTM outputs units, Relu (Li and Yuan) as our activation function and binary cross-entropy as our loss function.

BERT

BERT is a pre-trained contextual language representation model introduced by Google in 2018(Devlin et al. 2019). Unlike the previous models, BERT adapted transformer architecture and further develops encoder blocks into bidirectional structures, which allows the model to learn information from both previous and future context(Vaswani et al. 2017). Another advantage of BERT is, it is unsupervised trained on a large plain text corpus. This procedure frees users from requiring mass labeled data, thus alleviates the data scarcity issue. Moreover, pre-training on big text relieves the biased training set problem of the other context-aware word embedding methods.

There are reasons for BERT outperforming the other pre-trained methods. For example, in order to consider preceding and subsequent context, Bi-LSTMs train a left-to-right model as well as a right-to-left one(Schuster and Paliwal 1997). Yet this procedure somehow fails because it would let each word to indirectly “see itself” in a multi-layered condition. BERT introduced a ”Mask” procedure to tackle this problem, in which the model tries to predict what the masked word is from both directions. Furthermore, BERT also learns the relationship of different sentences with Next Sentence Prediction (NSP) tasks.

In conclusion, we employ BERT in this project, because it will mitigate our data scarcity concern, and also helps in decreasing the influence of the biased data issue of our target dataset, by providing a good pre-training procedure.

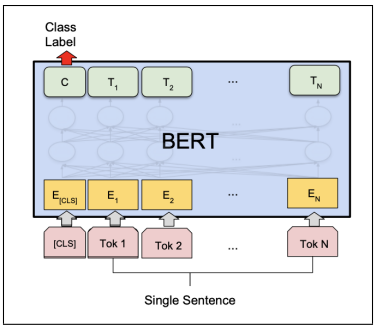

Figure 2: BERT on sentence classification task(Devlin et al. 2019).

In our project, we follow the pre-training and fine-tuning procedure suggested by the authors. Detailed descriptions are as follows.

1) Tokenization Because our data set is comparatively small, we use BERT-base-uncased module to break up our input entries into words representation.

2) BERT Embedding After tokenization, we convert our word vectors into a format that BERT understands, which contains token embedding, segment embedding, and position embedding.

3) Utilize Pre-trained BERT and Fine Tuning

In this stage, we first download BERT model from the Google official Github repository "google-research/bert". Then, to adapt the original binary classification job into our multi-label classification task, we make the following modification:

- use sigmoid instead of softmax in the last layer, because sigmoid function outputs continuous probability mapping, and softmax outputs one-hot encoding.

- change error measure function to "sigmoid cross entropy with logits". After accustoming the original model, we train the last layer for classification and then apply fine-tuning.

Experiment

In this project, we choose Kaggle toxic comments classification challenge (TCCC) as our text classification task. Conversation AI team and Google planned to construct a clean online conversation environment then they found several human raters to label more than 150k comments from Wikipedia talk page(Mohammad). This is a multi-label classification problem. The difference between multi-label and multi-class tasks is that the classes are mutually exclusive in multi-class problems, but an entry can be matched with several labels at the same time. In our project, all entries are labeled with 6 categories: toxic, severe toxic, obscene, threat, insult and identity hate. All the training set and test set are stored as csv format and the size of training set and test set are both more than 150K comments.

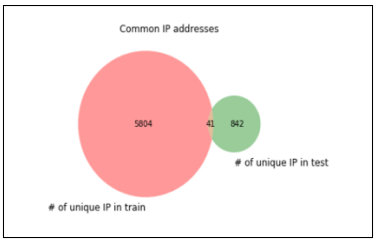

Figure 3: There are 41 comments from same IP address.

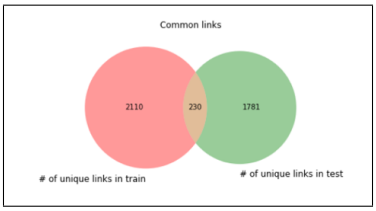

Figure 4: There are 290 comments containing same links.

Leaky Feature

In order to clean the dataset, we need to generate some new features from comments to check whether we should clean those contents regarding to these new features. In order to avoid data leakage, we generate three new features which is common of IP address, Common Links and Common usernames. Then we check whether there are some comments coming from the same IP, Links or username which means there might be a data leakage problem. The data analysis results are shown in figure 3, 4 and 5.

Indirect Features

From the analysis from the leaky features, we can discover that links and IP address can lead to data leakage since both training set and test set have many common information. Furthermore, although only one common username appear in both of the training set and the set, we still remove it on the safe side since only 157 and 81 comments containing username in training set and test set. Finally, we remove IP address and Link from each comment.

In this part, we defined some statistical features as signage to enrich our models. Our preference is to check whether a comment is toxic or not has some connections with these indirect features including number of unique words (unique), number of title words (title), number of upper words (up-

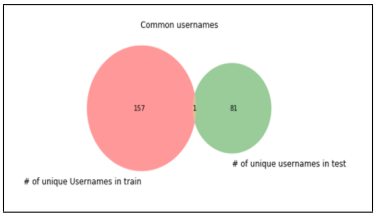

Figure 5: There are only one comment coming from the same username.

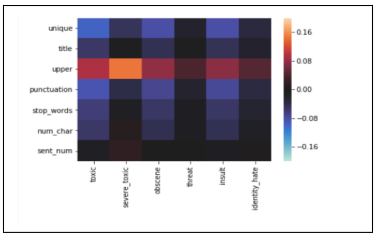

Figure 6: Number of unique words, number of upper words, number of stop words and number of char have different correlation value between the 6 types of toxic.

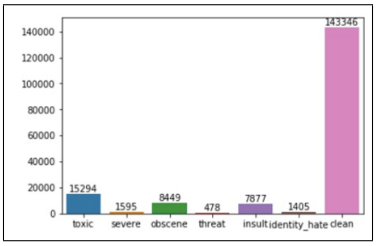

Figure 7: Most of the comments are clean.

per), number of punctuation (punctuation), number of stop words(stop_words), number of char(num_char) and number of sentence in one comment(sent_num). As a result, from correlation results from Figure 6 we decided to keep the stop words, unique words and upper words since these words might be some useful features to enhance our model.

Imbalanced Data

We count the number of comments in each of the 6 types of toxicity beside the clean comment. Then we found that most of the comments are clean with 143346 comments and only few of them are toxic one just as shown in Figure 7. Then we need treat this problem as an imbalanced classification problem. Whether balancing data will boosting or hurt the performance is an essential thing we need to confirm. So we need to use EDA to balancing the data to check the results.

Corpus Cleaning

Before we started to train our model, we need to clean our training set and test set. We performed the cleaning task from the following aspect:

- Acronym: We developed an apostrophe lookup dictionary which including all the possible apostrophe replacement scenario. Then we use it to standardize our corpus.
- Spelling Correction: We correct the spelling of some words and kept the case of the original words.
- Special replacement: In this part, we remove emoji, non-English words and some symbols.
- Punctuation abolishment : We decide to remove all the punctuation although the number of punctuation is a promising feature. We wish to use word vector to be our direct features then there is no need to keep punctuation. However, stop words are appropriate according to our analysis above then we decide to keep them.
- Leaky Feature Removement: It is essential to remove all the leaky features including IP address, Links and username. In such way, this removement method can help avoid overfitting due to data leakage.

Embedding Files	Embedding Size
Word2vec from Google News	300
Glove from Stanford university	50
fastText from Facebook AI team	300

Table 1: The dimensional information of these three different word embedding in our experiments.

Pre-trained Word Embedding

Initializing pre-trained word embedding has been a solid solution to boosting performance on text classification problem (Abadi et al. 2016). In this project, we use three various public word embedding files including word2vec(Pennington, Socher, and Manning 2014) trained from Google News, fastText (Liao et al. 2016) created by Facebook AI team and Glove embedding(Rong 2016) from Stanford university.

Data Augmentation by Improved EDA

Since we have removed α and number of augmentation, we just augmented all the training set and the size training set increased to 757929 samples. Then we keep the ratio between training set and validation set to be 95:5.

Training Model and Metrics

We run experiments on two-dimensional textCNN(Xiong et al. 2018) and Bidirectional LSTM(Cheng, Dong, and Lapata 2016) with Keras(Schoenauer-Sebag, Schoenauer, and Sebag 2017) framework, which can help construct and evaluation various model without implementing complex algorithm by hand. For embedding layer, we converted our corpus into word embedding format with the help of our public embedding files.

For textCNN, we use four 2D convolutional layers and 2D max-pooling layers. Number of filter is 32 and filter size for each of the four layers is 1,2,3,5 respectively. Elu function to be the activation function within each layer. Then we use 6 sigmoid function in our output layer and use Adam optimizer to dynamically adjust the learning rate depending on the first moment estimation and second prediction of the loss function gradient (Wang et al. 2016). Our loss function for textCNN is binary cross-entropy.

For bidirectional LSTM, the output dimension in the LSTM layer has been defined as 60.Then we use ReLu as our activation function. The dropout rate for the first operation of our inputs is 0.1 and the same value as the dropout rate for the application of the recurrent kernel. Then the optimizer and loss function are the same as that of textCNN. For the training phase, we set batch size to be 265 and number

	TextCnn (word2vec)	TextCnn (Glove)	TextCnn (fastText)	Bi-LSTM (word2vec)	Bi-LSTM (Glove)	Bi-LSTM (fastText)	BERT
Original Data	97.74	97.99	98.03	98.02	98.11	98.62	99.76
Improved EDA	98.29	98.32	98.51	98.48	98.70	99.01	99.87
Original EDA with Balanced Data	92.85	95.43	95.47	93.43	94.27	94.77	99.75

Table 2: Overall average accuracy for all the benchmarks.

of epoch to be 3.Then the proportion between training size and validation set is keeping in 0.95.

For BERT, we use BERT-base-uncased module to do tokenization. We modified the original model to adapt our multi-label classification task, then use the same architecture and parameter set when firstly training. After that, we also perform fine-tuning on the original parameter set to further adapt our dataset.

Finally, we use average accuracy between 6 types of toxicity as our performance metrics.

Results and Analysis

In this section, we show our evaluation results about this text classification task with or without improved EDA which are running on our models. Furthermore, we analyze the results to give some hints when handling data augmentation problem.

Overall Performance

This work implements six models, textCNN with Glove, textCNN with word2vec, textCNN with fastText, Bidirectional LSTM with Glove, Bidirectional LSTM with word2vec,Bidirectional LSTM with fastText and Bert Transformer. Table 2 shows the average accuracy as the performance (%) for our models running with improved EDA, original EDA and original EDA with balanced data.

Effect of improved EDA

Hopefully, we can discover that improved model works best among all 7 methods in our experiments. The average accuracy even increased to 99.87% using Bi-LSTM with fastText. Moreover, even though we have a large dataset with more than 700K records after data augmentation, the performance boosting will still keep in 0.5% in each of the model we tried in this experiments. As a result, we actually break the limitation that EDA can only work well in small dataset.

Conclusion

This paper introduces an improved version for EDA to reduce the risk of meaning changed and make it work well on a large dataset with more than 700K records. In our experiments, we tried different models and apply them to with original data, with improved EDA and with improved EDA on a balanced data. We can discover that Bi-LSTM work better than textCNN and fastText works best among all the three embedding files. BERT performs best as it was pre-trained on colossal dataset from other sources. Then we can see improved EDA boost performance on any model we have tried in this project. Finally, we reveal that generate too many augmentation can hurt performance when balancing the whole dataset.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mane, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viegas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv:1603.04467 [cs]*. arXiv: 1603.04467.

Abou-Assaleh, T.; Cercone, N.; Keselj, V.; and Sweidan, R. Detection of New Malicious Code Using N-grams Signatures. 4.

Antoniou, A.; Storkey, A.; and Edwards, H. 2018. Data Augmentation Generative Adversarial Networks. *arXiv:1711.04340 [cs, stat]*. arXiv: 1711.04340.

Bérard, F. 2003. The Magic Table: Computer-Vision Based Augmentation of a Whiteboard for Creative Meetings. 8.

Cheng, J.; Dong, L.; and Lapata, M. 2016. Long Short-Term Memory-Networks for Machine Reading. *arXiv:1601.06733 [cs]*. arXiv: 1601.06733.

Chiu, J. P., and Nichols, E. 2016. Named Entity Recognition with Bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics* 4:357–370.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*. arXiv: 1810.04805.

Edunov, S.; Ott, M.; Auli, M.; and Grangier, D. 2018. Understanding Back-Translation at Scale. *arXiv:1808.09381 [cs]*. arXiv: 1808.09381.

Frid-Adar, M.; Klang, E.; Amitai, M.; Goldberger, J.; and Greenspan, H. 2018. Synthetic data augmentation using GAN for improved liver lesion classification. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, 289–293. Washington, DC: IEEE.

Grau, S.; Sanchis, E.; Castro, M. J.; and Vilar, D. Dialogue act classication using a Bayesian approach. 5.

Graves, A.; Jaitly, N.; and Mohamed, A.-r. 2013. Hybrid speech recognition with Deep Bidirectional LSTM. In *2013*

IEEE Workshop on Automatic Speech Recognition and Understanding, 273–278. Olomouc, Czech Republic: IEEE.

Hu, M., and Liu, B. Mining Opinion Features in Customer Reviews. 6.

Huang, Z.; Xu, W.; and Yu, K. 2015. Bidirectional LSTM-CRF Models for Sequence Tagging. *arXiv:1508.01991 [cs]*. arXiv: 1508.01991.

Jindal, N. Mining Comparative Sentences and Relations. 6.

Kim, Y. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1746–1751. Doha, Qatar: Association for Computational Linguistics.

Li, X., and Roth, D. 2006. Learning question classifiers: the role of semantic information. *Natural Language Engineering* 12(3):229–249.

Li, Y., and Yuan, Y. Convergence Analysis of Two-layer Neural Networks with ReLU Activation. 11.

Liao, M.; Shi, B.; Bai, X.; Wang, X.; and Liu, W. 2016. TextBoxes: A Fast Text Detector with a Single Deep Neural Network. *arXiv:1611.06779 [cs]*. arXiv: 1611.06779.

Mohammad, F. Is preprocessing of text really worth your time for toxic comment classication? 7.

Pang, B., and Lee, L. 2004. A sentimental education: sentiment analysis using subjectivity summarization based on minimum cuts. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics - ACL '04*, 271–es. Barcelona, Spain: Association for Computational Linguistics.

Pennington, J.; Socher, R.; and Manning, C. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. Doha, Qatar: Association for Computational Linguistics.

Rong, X. 2016. word2vec Parameter Learning Explained. *arXiv:1411.2738 [cs]*. arXiv: 1411.2738.

Schoenauer-Sebag, A.; Schoenauer, M.; and Sebag, M. 2017. Stochastic Gradient Descent: Going As Fast As Possible But Not Faster. *arXiv:1709.01427 [cs, stat]*. arXiv: 1709.01427.

Schuster, M., and Paliwal, K. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45(11):2673–2681.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A.; and Potts, C. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. 12.

Stabek, A.; Brown, S.; and Watters, P. 2009. The Case for a Consistent Cyberscam Classification Framework (CCCF). In *2009 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*, 525–530. Brisbane, Australia: IEEE.

Taylor, L., and Nitschke, G. 2017. Improving Deep Learning using Generic Data Augmentation. *arXiv:1708.06020 [cs, stat]*. arXiv: 1708.06020.

Topkara, U.; Topkara, M.; and Atallah, M. J. The Hiding Virtues of Ambiguity: Quantiably Resilient Watermarking of Natural Language Text through Synonym Substitutions. 11.

van Aken, B.; Risch, J.; Krestel, R.; and Löser, A. 2018. Challenges for Toxic Comment Classification: An In-Depth Error Analysis. *arXiv:1809.07572 [cs]*. arXiv: 1809.07572.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need. *arXiv:1706.03762 [cs]*. arXiv: 1706.03762.

Wang, P.; Xu, B.; Xu, J.; Tian, G.; Liu, C.-L.; and Hao, H. 2016. Semantic expansion using word embedding clustering and convolutional neural network for improving short text classification. *Neurocomputing* 174:806–814.

Wang, T.; Qin, Z.; and Zhu, M. 2017. An ELU Network with Total Variation for Image Denoising. *arXiv:1708.04317 [cs]*. arXiv: 1708.04317.

Wei, J., and Zou, K. 2019. EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. *arXiv:1901.11196 [cs]*. arXiv: 1901.11196.

Wilson, T.; Wiebe, J.; and Hoffmann, P. Recognizing Contextual Polarity in Phrase-Level Sentiment Analysis. 8.

Xiong, W.; Ni'mah, I.; Huesca, J. M. G.; van Ipenburg, W.; Veldsink, J.; and Pechenizkiy, M. 2018. Looking Deeper into Deep Learning Model: Attribution-based Explanations of TextCNN. *arXiv:1811.03970 [cs, stat]*. arXiv: 1811.03970.

Zhang, X., and LeCun, Y. 2016. Text Understanding from Scratch. *arXiv:1502.01710 [cs]*. arXiv: 1502.01710.

Zhou, P.; Qi, Z.; Zheng, S.; and Xu, J. Text Classification Improved by Integrating Bidirectional LSTM with Two-dimensional Max Pooling. 11.