Word Embedding Annealing Using Sequence-to-sequence Model

Taehoon Kim and Jihoon Yang
Data Mining Research Laboratory
Department of Computer Science and Engineering
Sogang University
{taehoonkim, yangjh}@sogang.ac.kr

Abstract

We propose a new technique to improve text classification accuracy by annealing the word embedding layer using Seq2seq Neural Networks. Our model Sequence-to-convolution Neural Networks(Seq2CNN) consists of two blocks: Sequential Block that summarizes input texts and Convolution Block that classifies the original text to a label. We also present Gradual Weight Shift(GWS) method that stabilize training. GWS is applied to our model's loss function. We compared our model with word-based TextCNN trained with different data preprocessing methods. We obtained some improvement in classification accuracy over word-based TextCNN without any ensemble or data augmentation. Code is available at https://github.com/tgisaturday/Seq2CNN.

1 Introduction

Ever since humans began to record information in the form of text, it was necessary to classify and manage information in a certain category to store and retrieve information efficiently. This need encouraged many researchers to develop a good text classification technique that can assign predefined categories to various kinds of text document such as emails, news articles, reviews, or patents.

In commercial world, text classification techniques such as Naïve Bayes classifier[5], TFIDF[36], Support Vector Machines(SVM)[15] are already used in various fields including spam filtering, news categorization, and sentiment analysis. Recent development in deep neural networks[17, 38, 19, 37, 7] are also achieving excellent results in extracting information from a text and classifying it into certain classes.

As Convolutional Neural Networks(CNNs) achieved remarkable results in computer vision[31, 34, 11, 28], researchers also applied CNNs to text classification[17, 19, 37, 7] and showed excellent results. Training CNNs on top of pretrained word vectors[22, 16, 27] or character-level features[37, 7] with hyperparameter tuning, they could get similar or outperforming results compared to other text classification models.

Along with TextCNNs' remarkable performance in text classification, improving the quality of the word embedding became also important. In section 4.2, we show that performance of TextCNNs can be improved by annealing its word embedding layer with Seq2seq model that summarizes the input text. There are two ways to generate the summary of a text. One is extractive summarization, mere selection of a few existing sentences extracted from the source. The other is abstractive summarization, compressed paraphrasing of main contents of source, potentially using vocabulary unseen in the source. Both methods can change texts of various lengths into texts of fixed length still maintaining important features of source texts.

Preprint. Work in progress.

TextRank[21] is a graph-based ranking model for extractive text summarization. TextRank gives a ranking over all sentences in a text allowing it to extract very short summaries without any training corpora. TextRank is widely used in summarizing structured text like news articles.

Many researchers worked with Sequence-to-sequence Recurrent Neural Networks (Seq2seq RNNs)[33, 24] to model abstractive text summarization. Using attention mechanism[6] that allows neural networks to focus on different parts of their input, Seq2seq RNNs have been showing significant results in the task of abstractive summarization[29, 24].

In this paper, we introduce Sequence-to-convolution Neural Networks(Seq2CNN) model that consists of two blocks: Sequence Block and Convolution Block. Sequence Block based on Attentional Encoder-Decoder Recurrent Neural Networks[24] summarizes input texts and feeds them into Convolution Block. Convolution Block based on TextCNN[17] classifies input texts into certain classes. Both blocks share non-static word embedding layer, encouraging them to collaborate for performance improvement.

Simply connecting two blocks and train them with single end-to-end procedure cannot guarantee optimal results because Sequential Block doesn't generate proper summaries in early stages of training. To solve this problem, we also propose a new training scheme that gradually shifts from fine-tuning for summarization task to fine-tuning for classification task as training progresses. Our model is implemented with Tensorflow[1]. Code is available at https://github.com/tgisaturday/Seq2CNN.

2 Related Work

There was similar approach of text classification with summaries using Latent Semantic Analysis(LSA) as extractive summarization method[9]. They proposed a hybrid model for unlabeled document classification using SVM classifier with classification rules are generated using summaries of the training documents. Although we cannot directly compare the performance because of the domain difference in training data, we discuss performance of TextCNN trained with extractive summaries generated with TextRank[21].

Mnih et al.[23] came up with novel RNN models with visual attention that is capable of extracting information from an image or video by adaptively selecting a sequence of regions and this idea of using attention mechanism was successfully applied to machine translation by Bahdanau et al.[2]. We used Bahdanau Attention[2] to improve the performance of our Sequential Block.

Our approach to use Attentional Encoder-Decoder RNNs for abstractive summarization is closely related to Nallapati et al.[24] who were the first to use Seq2seq RNNs and Attention model for abstractive text summarization. Our Tensorflow[1] implementation of Sequential Block is very similar to Pan and Liu[20]. They used Annotated English Gigaword[25] dataset to train their model and achieved state-of-the-art results (as of June 2016).

Seq2seq models[33, 24] require training set of input text and corresponding example summary. None of datasets that we used for evaluation has any example summary. We used TextRank[21] to generate extractive summaries out of input texts and feed them to Sequential Block as example summary. We chose TextRank[21] because it can generate practical summaries even with short texts and doesn't require any training corpora.

The architecture of our Convolution Block is based on TextCNN model proposed by Kim[17]. Kim[17] was the first to use CNN in text classification. Using this model as a baseline, we applied Batch Normalization after each convolution layer and changed hyperparameters for optimization.

Our training scheme is closely related to the one proposed in Faster R-CNN paper[28]. Their scheme alternates between fine-tuning two different tasks. We also tried to train our model with training scheme that alternates between fine-tuning summarization task and classification task. However, this was not effective because summaries generated in early stages of training were filled with series of UNK (unknown word) tokens. Instead, our training scheme gradually changes focus of training from summary generation to text classification so that summaries generated in early stages of training don't lower the classification accuracy of Convolution Block.

3 Seq2CNN Model

Figure 1 depicts the overall structure of Seq2CNN model.

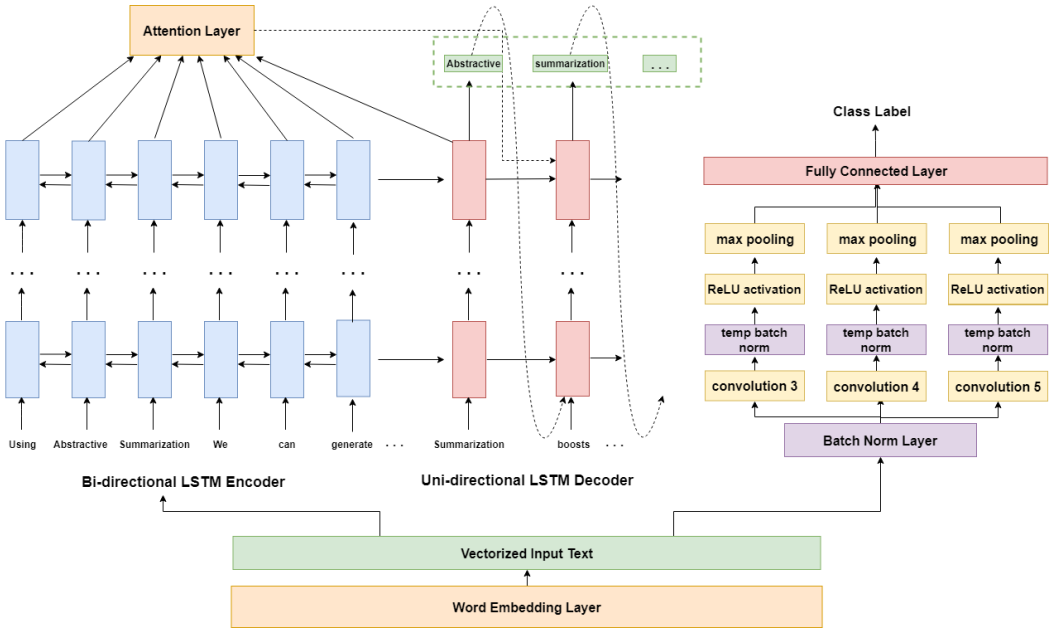

Figure 1: Overview of our Seq2CNN model that consists of Sequential Block(left) and Convolution Block(right). Both blocks interact with Word Embedding Layer to get vectorized representation of words used in summarization and classification tasks.

3.1 Sequential Block

Our baseline model of Sequential Block corresponds to the Attentional Encoder-Decoder RNN model used in Nallapati et al.[24] which encodes a source sentence into a fixed-length vector from which decoder generates abstractive summaries. The encoder consists of Bidirectional RNN[30] and decoder consists of Uni-directional RNN[10]. We used Long Short-Term Memory RNN[13] with 128 hidden units for encoder and decoder and Bahdanau Attention[2] for attention mechanism. We also inserted Dropout[32] modules between LSTM layers to regularize.

The forward LSTM of encoder reads the input sequence as it is ordered, and the backward LSTM reads the sequence in the reverse order. In this way, fixed-length vector from encoder contains the summaries of both preceding words and the following words. With the help of attention mechanism, decoder decides parts of the source sentence to pay attention to and focus only on the vectors that are essential for summarization.

3.2 Convolution Block

The structure of our Convolution Block is based on TextCNN model proposed by Kim[17] which gets $n \times k$ vectorized representation of text where n is the number of words inside the text and k is the dimension size of word embedding. Each filter windows with varying sizes(h) extracts one feature by performing $h \times k$ convolution operation over input and apply max-over-time pooling operation. The model uses multiple filters to multiple features. These features are passed to a fully-connected softmax layer whose output is the probability distribution over labels.

Convolution Block gets vectorized representation of input texts. We used rectified linear units(ReLU)[35] for non-linear activation function and filter windows of 3, 4, 5 with 32 filters each. We applied Batch Normalization[14] after each convolution layer. Batch Normalization[14] accelerates training by reducing internal covariate shift[14], the change in the distribution of network activations due to the change in network parameters during the training. We also applied batch

normalization on vectorized representation of summaries to stabilize entire training procedure by reducing internal covariate shift between Sequential block and Convolution block. For regularization, we inserted Dropout[32] module between max-pooling layer and fully-connected layer.

3.3 Word Embedding Block

Word Embedding Block consists of word embedding layer which stores vectorized representations of each word and vocabulary lookup table that maps each word with corresponding vector representation. To make vocabulary dictionary, we extracted 20,000 to 30,000 words from training data with minimum word frequency(f), excluding words that have appeared less than f times. Our word embedding layer is non-static and fine-tuned via back-propagation. In our implementation, we set word embedding dimension to 100.

3.4 Loss Function

The main objective of Seq2CNN is to classify texts of various lengths without losing important features of original context. For feature extraction, we use abstractive summarization method using Sequential Block. Although our model is mainly focused on classification, quality of summary must be guaranteed for Convolution Block to successfully perform classification task.

Taking everything into consideration, we trained our model to minimize an objective function which is weighted sum of losses in classification and summarization. Our loss function for a text is defined as:

$$L(\{p_{y_i}, y_i^*\}, \{t_i, t_i^*, l_i\}) \;=\; \frac{1}{N}\left[L_{cls}(p_{y_i}, y_i^*) \;+\; \lambda L_{sum}(t_i, t_i^*, l_i)\right] \tag{1}$$

$$L_{sum}(t_i, t_i^*, l_i) \;=\; \frac{1}{l_i}\sum_{j}^{l_i} L_{vocab_j}(p_{w_j}, w_j^*) \tag{2}$$

In (1), i is the index of a text in a mini-batch of size N and p_{y_i} is the predicted probability of text i. classified as ground truth label y_i^*. The classification loss L_{cls} is cross-entropy loss over classification classes. The summarization loss L_{sum} is sequence loss between the summary output of Sequential Block t_i with length l_i and summary example t_i^*.

In (2), p_{w_j} is the predicted probability of the jth word w_j in generated summaryto match with the jth word of summary example w_{j^*}. We defined sequence loss as the average of the vocabulary losses L_{vocab}s in t_i. The vocabulary loss of jth word L_{vocab_j} is cross-entropy loss over vocabulary in dictionary stored in Word Embedding Block.

Total loss is weighted sum of L_{cls} and L_{sum} with a balancing weight λ, normalized with N. In our implementation, we used multi-class softmax layer to get p_{y_i} and p_{w_j}. We normalized our loss with the mini-batch size. We applied Gradual Weight Shift to λ. We explained more about Gradual Weight Shift in section 3.5.

3.5 Gradual Weight Shift

In our implementation of Seq2CNN, we train Sequential Block and Convolution Block end-to-end by back-propagation and gradient optimizer using loss function defined in 3.4. This training scheme fine-tunes the model and reduces training time compared to training each model independently. However, using a constant value (>1.0) for balancing weight λ caused sudden drops of validation accuracy in later stages of training (~20 epochs). We found the main cause of this phenomenon in Sequential Block.

In earlier stages of training (~10 epochs), Sequential Block does not generate practical summaries and omits UNK tokens instead. Huge difference in quality of summary throughout the training hinders the optimization of Convolution Block. Using larger λ (>1.5) solves this problem a little bit by giving more weight to L_{sum}, making Sequential Block to converge faster.

Using larger λ gives more weight to L_{sum} than L_{cls}, leading the model to focus on optimization of Sequential Block until the end. Since our model is designed for classification, we gradually shifted

weight from L_{sum} to L_{cls} by exponentially decaying λ throughout time. Our exponential decay function for λ is defined as:

$$\lambda_t \ = \ \lambda_0 \delta^t \tag{3}$$

where λ_0 is initial value of lambda, δ is decay rate, and t is current time step. We could stabilize the training of our model and achieve higher test accuracy by applying Gradual Weight Shift to our loss function defined in section 3.4. We explain more about the result in section 4.3.

3.6 Sharing Word Embedding for Summarization and Classification

We designed our model to share word embedding for summarization and classification. When back-propagation happens, Sequential Block tries to update word embedding layer in direction of minimizing sequence loss. On the other hand, Convolution Block tries to update word embedding layer in direction of minimizing classification loss. This anneals the word embedding layer.

3.7 Optimization

In our implementation, we trained our model end-to-end by back-propagation and stochastic gradient descent(SGD)[4] using Adam optimizer[18] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 0.1$. We used learning rate of 0.001, decayed every epoch using an exponential rate of 0.95. Dropout rate for Dropout[32] modules is 0.7. We also used gradient clipping[26] with gradient norm limited to 5. We set loss balancing weight λ to 1.0 when training AG's News and 2.0 for other datasets.

We initialized convolution layers using He Normal[12] initializer and fully-connected layer using Xavier[8] initializer. All other weights were initialized with random values from a uniform distribution with minimum of -1 and maximum of 1. The implementation is done using Tensorflow[1]. We trained our model using single NVIDIA Titan V GPU with mini-batch size 256. It took 3 hours(AG's News) to 1 day(Yahoo Answers) for training. We didn't used any ensemble or data augmentation techniques Detailed information about datasets are given in section 4.1.

4 Experiments

We evaluated the performance of our model by comparing with basic TextCNN[17]. We defined basic TextCNN used in experiments as Vanilla CNN. We used the same hyperparameters(filter windows of 3, 4, 5 with 32 filters each) for Vanilla CNN and Convolution Block in Seq2CNN. We trained Vanilla CNN with three different data preprocessing methods: **full-text, crop&pad, summarize**. We defined the length of text as the number of words in each text.

full-text This is default data preprocessing method. We just removed unnecessary characters and stopwords from the input texts and padded them into fixed length with PAD token. Here, fixed length is the maximum length of text in each dataset. Same preprocessing method is also applied before **crop&pad** and **summarize**.

crop&pad Using fixed length as hyperparameter(numbers inside bracket in Table 2), we cropped each text into fixed length. For example, if a text is longer than 20 words, we only used 20 words starting from the front. Texts shorter than fixed length is padded with PAD token.

summarize Instead of cropping each sentence, we generated extractive summary of input text using TextRank[21] and processed the summary with **crop&pad**. Same method was used to generate summary example used in training Sequential Block.

4.1 Datasets

We evaluated our model on three different datasets: AG's News, DBPedia, and Yahoo Answers[37]. To offer fair evaluation on the performance of Sequential Block in Seq2CNN, we removed input texts shorter than 50 words(maximum fixed length value in Table 2), changing number of training samples in DBPedia and Yahoo Answers. The number of training samples in AG' News is the same as the original. We didn't apply any changes to test samples for fair comparison with best published results. We also limited the size of vocabulary into 20,000 to 30,000 with minimum word frequency(f) in

Table 1: Statisics of datasets. Vocabulary size is number of words used to train the model. Min Freq is minimum word frequency f used to decide vocabulary size of each dataset.

Datasets	Classes	Training Set	Test Set	Vocabulary Size	Min Freq
AG's News	4	120,000	7,600	26,543	3
DBPedia	14	140,000	70,000	23,371	15
Yahoo Answers	10	150,000	60,000	28,451	15

Table 1. Words in test samples are not included in vocabulary dictionary of Word Embedding Block. None of the datasets contains summary examples to train Seq2CNN model so we generated summary examples using TextRank[21]. We didn't feed any summary samples while evaluating Seq2CNN with test samples.[1] Detailed statistics of each dataset is given in Table 1.

4.2 Text Classification

Table 2: Classification results of all models. Numbers are test accuracy in percentage. "Vanilla CNN" is basic TextCNN[17] model and "Seq2CNN" is our model. "Full" stands for "full-text", "Crop" stands for "crop&pad", and "Sum" stands for "summarize". We labeled the best result of Vanilla CNN in blue and worst result in red. Best result of Seq2CNN is labeled in green.

Model	AG's News	DBPedia	Yahoo Answers
Vanilla CNN Full	89.28	96.39	54.37
Vanilla CNN Crop(20)	89.42	96.69	54.28
Vanilla CNN Crop(50)	89.48	96.38	54.83
Vanilla CNN Sum(20)	89.59	96.84	53.98
Vanilla CNN Sum(50)	89.67	96.89	54.69
Seq2CNN(20)	90.18	97.07	55.06
Seq2CNN(50)	90.36	97.23	55.39

Extractive Text Classification Table 2 shows classification results of Seq2CNN and Vanilia CNN models. We first evaluated data preprocessing methods for Vanilla CNN with different fixed length sizes. We labeled the best result in blue and worst result in red. There was not a single data preprocessing method that derived best performance for all datasets. Summarization with TextRank[21] tends to work well in most of the cases.

Abstractive Text Classification Our Seq2CNN model outperformed other models in all cases bringing average 1% growth compared to Vanilla CNN trained without any data preprocessing(Vanilla CNN Full). Best published result for AG's News using TextCNN[37] is 90.09% with pretrained word2vec[22] embedding and data augmentation technique[37] using thesaurus. Our model achieved competitive result on AG's News dataset without any pretrained word embedding or data augmentation technique. We cannot directly compare other results due to the changes that we explained in section 4.1.

4.3 Gradual Weight Shift

Optimization In sequence loss curve of Figure 2, the sequence loss of the model trained without GWS converges smoothly in the early stage of training, but starts to fluctuate after 2,000 steps. This also effects the total loss curve, unstabilizing the training of the model. In contrast, loss curves of the model with GWS converges smoothly until the end.

Classification Performance We evaluated the performance of Gradual Weight Shift with AG's News Dataset. In Table 4, Seq2CNN model trained with GWS achieved better results compared to the same model trained without GWS. Although Seq2CNN model without GWS performed better

[1] In our implementation with Tensorflow, we used greedy embedding helper instead of training helper for inference layer.

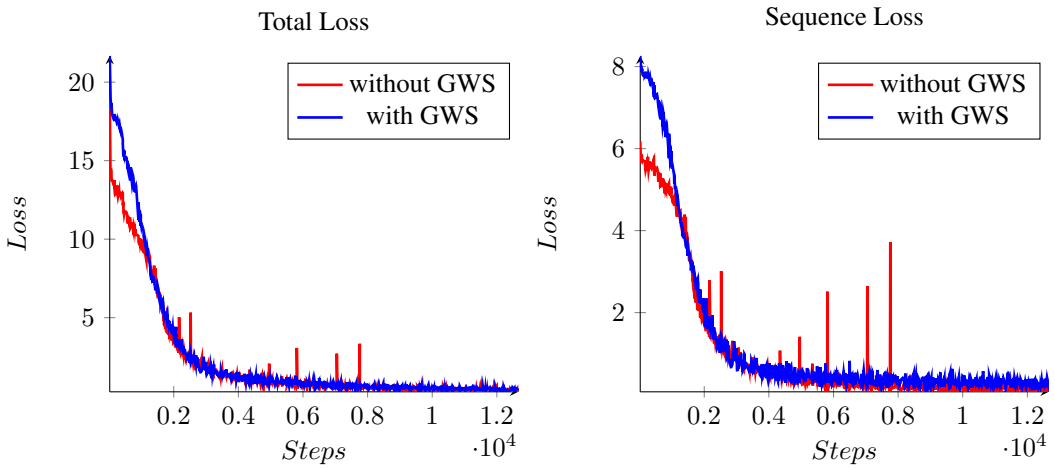

Figure 2: Total loss curve(left) and sequence loss curve(right) on the **AG's News Dataset** with and without Gradual Weight Shift(GWS).

Table 3: Classification results on the **AG's News Dataset** with and without Gradual Weight Shift(GWS). Best result using Vanilla CNN is also included for comparison.

Model	Accuracy
Vanilla CNN	89.67
Seq2CNN without GWS(20)	89.75
Seq2CNN without GWS(50)	89.87
Seq2CNN with GWS(20)	90.18
Seq2CNN with GWS(50)	90.36

than any other Vanilla CNN models, it could not outperform previous best published result of Zhang et al.[37].

5 Conclusion

We have proposed Sequence-to-Convolution Neural Networks (Seq2CNN) for efficient and accurate text classification. We also presented a new training theme for our model using Gradual Weight Shift(GWS), which can be applied to other models with multi-task loss function by changing number of balancing weights.

We adopted general sequence loss function which uses predicted probability of jth word w_j in generated summary matches jth word of summary example w_{j^*} to calculate vocabulary losses. However, we think the sequence loss cannot be evaluated accurately by comparing only the words in the same position. Bahdanau et al.[3] suggested specialized surrogate losses for Encoder-Decoder models often used for sequence prediction tasks and brought significant performance improvements.

We also didn't use any pretrained word embeddings to initialize the Word Embedding Block. Previous results on word-based TextCNN [17, 37, 19] suggests that initializing embedding layers with pre-trained word vectors such as word2vec[22], FastText[16], or GloVe[27] helps improves performances of models.

In the future, we are planning to improve our basic word embedding annealing technique for better results and wider usages.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow,

Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[3] Dzmitry Bahdanau, Dmitriy Serdyuk, Philemon Brakel, Nan Rosemary Ke, Jan Chorowski, Aaron C. Courville, and Yoshua Bengio. Task loss estimation for sequence prediction. *CoRR*, abs/1511.06456, 2015.

[4] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186, Heidelberg, 2010. Physica-Verlag HD.

[5] Jingnian Chen, Houkuan Huang, Shengfeng Tian, and Youli Qu. Feature selection for text classification with naïve bayes. *Expert Systems with Applications*, 36(3, Part 1):5432 – 5435, 2009.

[6] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 577–585. Curran Associates, Inc., 2015.

[7] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann LeCun. Very deep convolutional networks for natural language processing. *CoRR*, abs/1606.01781, 2016.

[8] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[9] Marlene Grace, U Rajasekhar, Vijayapal Reddy, and A Vinaya Babu. Text classification with summaries generated using latent semantic analysis. *IJRSAE*, 2, 12 2014.

[10] A. Graves, A. r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[13] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(9):1735–1780, November 1997.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015.

[15] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Machine Learning: ECML-98*, pages 137–142, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[16] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759, 2016.

[17] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[19] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. Recurrent convolutional neural networks for text classification, 2015.

[20] Peter Liu and Xin Pan. Text summarization with tensorflow, 2016. Software available from tensorflow.org.

[21] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into texts. In Dekang Lin and Dekai Wu, editors, *Proceedings of EMNLP 2004*, pages 404–411, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[22] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[23] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. Recurrent models of visual attention. *CoRR*, abs/1406.6247, 2014.

[24] Ramesh Nallapati, Bing Xiang, and Bowen Zhou. Sequence-to-sequence rnns for text summarization. *CoRR*, abs/1602.06023, 2016.

[25] Courtney Napoles, Matthew Gormley, and Benjamin Van Durme. Annotated gigaword. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*, AKBC-WEKEX '12, pages 95–100, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[26] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.

[27] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[28] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[29] Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. *CoRR*, abs/1509.00685, 2015.

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

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

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

[33] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[34] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[35] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015.

[36] Zhang Yun-tao, Gong Ling, and Wang Yong-cheng. An improved tf-idf approach for text classification. *Journal of Zhejiang University-SCIENCE A*, 6(1):49–55, Aug 2005.

[37] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626, 2015.

[38] Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. A C-LSTM neural network for text classification. *CoRR*, abs/1511.08630, 2015.