LeafNATS: An Open-Source Toolkit and Live Demo System for Neural Abstractive Text Summarization

Tian Shi
Virginia Tech
tshi@vt.edu

Ping Wang
Virginia Tech
ping@vt.edu

Chandan K. Reddy
Virginia Tech
reddy@cs.vt.edu

Abstract

Neural abstractive text summarization (NATS) has received a lot of attention in the past few years from both industry and academia. In this paper, we introduce an open-source toolkit, namely LeafNATS, for training and evaluation of different sequence-to-sequence based models for the NATS task, and for deploying the pre-trained models to real-world applications. The toolkit is modularized and extensible in addition to maintaining competitive performance in the NATS task. A live news blogging system has also been implemented to demonstrate how these models can aid blog/news editors by providing them suggestions of headlines and summaries of their articles.

1 Introduction

Being one of the prominent natural language generation tasks, neural abstractive text summarization (NATS) has gained a lot of popularity (Rush et al., 2015; See et al., 2017; Paulus et al., 2017). Different from extractive text summarization (Gambhir and Gupta, 2017; Nallapati et al., 2017; Verma and Lee, 2017), NATS relies on modern deep learning models, particularly sequence-to-sequence (Seq2Seq) models, to generate words from a vocabulary based on the representations/features of source documents (Rush et al., 2015; Nallapati et al., 2016), so that it has the ability to generate high-quality summaries that are verbally innovative and can also easily incorporate external knowledge (See et al., 2017). Many NATS models have achieved better performance in terms of the commonly used evaluation measures (such as *ROUGE* (Lin, 2004) score) compared to extractive text summarization approaches (Paulus et al., 2017; Celikyilmaz et al., 2018; Gehrmann et al., 2018).

We recently provided a comprehensive survey of the Seq2Seq models (Shi et al., 2018), including their network structures, parameter inference methods, and decoding/generation approaches, for the task of abstractive text summarization. A variety of NATS models share many common properties and some of the key techniques are widely used to produce well-formed and human-readable summaries that are inferred from source articles, such as encoder-decoder framework (Sutskever et al., 2014), word embeddings (Mikolov et al., 2013), attention mechanism (Bahdanau et al., 2014), pointing mechanism (Vinyals et al., 2015) and beam-search algorithm (Rush et al., 2015). Many of these features have also found applications in other language generation tasks, such as machine translation (Bahdanau et al., 2014) and dialog systems (Serban et al., 2016). In addition, other techniques that can also be shared across different tasks include training strategies (Goodfellow et al., 2014; Keneshloo et al., 2018; Ranzato et al., 2015), data pre-processing, results post-processing and model evaluation. Therefore, having an open-source toolbox that modularizes different network components and unifies the learning framework for each training strategy can benefit researchers in language generation from various aspects, including efficiently implementing new models and generalizing existing models to different tasks.

In the past few years, different toolkits have been developed to achieve this goal. Some of them were designed specifically for a single task, such as ParlAI (Miller et al., 2017) for dialog research, and some have been further extended to other tasks. For example, OpenNMT (Klein et al., 2017) and XNMT (Neubig et al., 2018) are primarily for neural machine translation (NMT), but have been applied to other areas. The bottom-up attention model (Gehrmann et al., 2018), which

Proceedings of NAACL-HLT 2019: Demonstrations, pages 66–71
Minneapolis, Minnesota, June 2 - June 7, 2019.

has achieved state-of-the-art performance for abstractive text summarization, is implemented in OpenNMT. There are also several other general purpose language generation packages, such as Texar (Hu et al., 2018). Compared with these toolkits, LeafNATS is specifically designed for NATS research, but can also be adapted to other tasks. In this toolkit, we implement an end-to-end training framework that can minimize the effort in writing codes for training/evaluation procedures, so that users can focus on building models and pipelines. This framework also makes it easier for the users to transfer pre-trained parameters of user-specified modules to newly built models.

In addition to the learning framework, we have also developed a web application, which is driven by databases, web services and NATS models, to show a demo of deploying a new NATS idea to a real-life application using LeafNATS. Such an application can help front-end users (e.g., blog/news authors and editors) by providing suggestions of headlines and summaries for their articles.

The rest of this paper is organized as follows: Section 2 introduces the structure and design of LeafNATS learning framework. In Section 3, we describe the architecture of the live system demo. Based on the request of the system, we propose and implement a new model using LeafNATS for headline and summary generation. We conclude this paper in Section 4.

2 LeafNATS Toolkit[1]

In this section, we introduce the structure and design of LeafNATS toolkit, which is built upon the lower level deep learning platform – Pytorch (Paszke et al., 2017). As shown in Fig. 1, it consists of four main components, i.e., engines, modules, data and tools and playground.

Engines: In LeafNATS, an engine represents a training algorithm. For example, end-to-end training (See et al., 2017) and adversarial training (Goodfellow et al., 2014) are two different training frameworks. Therefore, we need to develop two different engines for them.

Specifically for LeafNATS, we implement a task-independent end-to-end training engine for NATS, but it can also be adapted to other NLP tasks, such as NMT, question-answering, sentiment classification, etc. The engine uses abstract data, models, pipelines, and loss functions

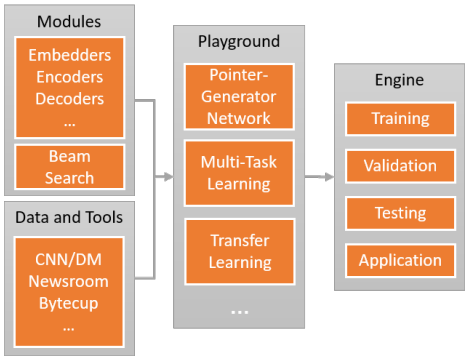

Figure 1: The framework of LeafNATS toolkit.

to build procedures of training, validation, testing/evaluation and application, respectively, so that they can be completely reused when implementing a new model. For example, these procedures include saving/loading check-point files during training, selecting N-best models during validation, and using the best model for generation during testing, etc. Another feature of this engine is that it allows users to specify part of a neural network to train and reuse parameters from other models, which is convenient for transfer learning.

Modules: Modules are the basic building blocks of different models. In LeafNATS, we provide ready-to-use modules for constructing recurrent neural network (RNN)-based sequence-to-sequence (Seq2Seq) models for NATS, e.g., pointer-generator network (See et al., 2017). These modules include embedder, RNN encoder, attention (Luong et al., 2015), temporal attention (Nallapati et al., 2016), attention on decoder (Paulus et al., 2017) and others. We also use these basic modules to assemble a pointer-generator decoder module and the corresponding beam search algorithms. The embedder can also be used to realize the embedding-weights sharing mechanism (Paulus et al., 2017).

Data and Tools: Different models in LeafNATS are tested on three datasets (see Table 1), namely, CNN/Daily Mail (CNN/DM) (Hermann et al., 2015), Newsroom (Grusky et al., 2018) and Bytecup[2]. The pre-processed CNN/DM data is available online[3]. Here, we provide tools to preprocess the last two datasets. Data modules are used to prepare the input data for mini-batch optimization.

Playground: With the engine and modules, we can develop different models by just assembling

[1] https://github.com/tshi04/LeafNATS

[2] https://biendata.com/competition/bytecup2018/

[3] https://github.com/JafferWilson/Process-Data-of-CNN-DailyMail

Dataset	Train	Validation	Test
CNN/DM	287,227	13,368	11,490
Newsroom	992,985	108,612	108,655
Bytecup	892,734	111,592	111,592

Table 1: Basic statistics of the datasets used.

these modules and building pipelines in playground. We re-implement different models in the NATS toolkit (Shi et al., 2018) to this framework. The performance (ROUGE scores (Lin, 2004)) of the pointer-generator model on different datasets has been reported in Table 2, where we find that most of the results are better than our previous implementations (Shi et al., 2018) due to some minor changes to the neural network.

Dataset	Model	R-1	R-2	R-L
Newsroom-S	Pointer-Generator	39.91	28.38	36.87
Newsroom-H	Pointer-Generator	27.11	12.48	25.47
CNN/DM	Pointer-Generator	37.02	15.97	34.18
	+coverage	39.26	17.21	36.16
Bytecup	Pointer-Generator	40.50	24.57	37.63

Table 2: Performance of our implemented pointer-generator network on different datasets. Newsroom-S and -H represent Newsroom summary and headline datasets, respectively.

3 A Live System Demonstration[4]

In this section, we present a real-world web application of the abstractive text summarization models, which can help front-end users to write headlines and summaries for their articles/posts. We will first discuss the architecture of the system, and then, provide more details of the front-end design and a new model built by LeafNATS that makes automatic summarization and headline generation possible.

3.1 Architecture

This is a news/blog website, which allows people to read, duplicate, edit, post, delete and comment articles. It is driven by web-services, databases and our NATS models. This web application is developed with PHP, HTML/CSS, and jQuery following the concept of Model-View-Controller (see Fig. 2).

In this framework, when people interact with the front-end views, they send HTML requests to controllers that can manipulate models. Then, the views will be changed with the updated information. For example, in NATS, we first write an article in a text-area. Then, this article along with

[4] http://dmkdt3.cs.vt.edu/leafNATS

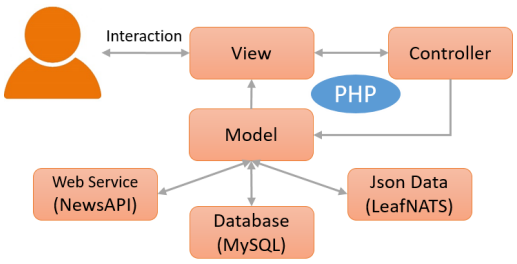

Figure 2: The architecture of the live system.

the summarization request will be sent to the controller via jQuery Ajax call. The controller communicates with our NATS models asynchronously via JSON format data. Finally, generated headlines and summaries are shown in the view.

3.2 Design of Frontend

Fig. 4 presents the front-end design of our web application for creating a new post, where labels represent the sequence of actions. In this website, an author can first click on "New Post" (step 1) to bring a new post view. Then, he/she can write content of an article in the corresponding text-area (step 2) without specifying it's headline and highlights, i.e., summary. By clicking "NATS" button (step 3) and waiting for a few seconds, he/she will see the generated headlines and highlights for the article in a new tab on the right hand side of the screen. Here, each of the buttons in gray color denotes the resource of the training data. For example, "Bytecup" means the model is trained with Bytecup headline generation dataset. The tokenized article content is shown in the bottom. Apart from plain-text headlines and highlights, our system also enables users to get a visual understanding of how each word is generated via attention weights (Luong et al., 2015). When placing the mouse tracker (step 4) on any token in the headlines or highlights, related content in the article will be labeled with red color. If the author would like to use one of the suggestions, he/she can click on the gray button (step 5) to add it to the text-area on the left hand side and edit it. Finally, he/she can click "Post" (step 6) to post the article.

3.3 The Proposed Model

As shown in the Fig. 3, our system can suggest to the users two headlines (based on Newsroom headline and Bytecup datasets) and summaries (based on Newsroom summary and CNN/DM datasets). They are treated as four tasks in this section. To achieve this goal, we use the mod-

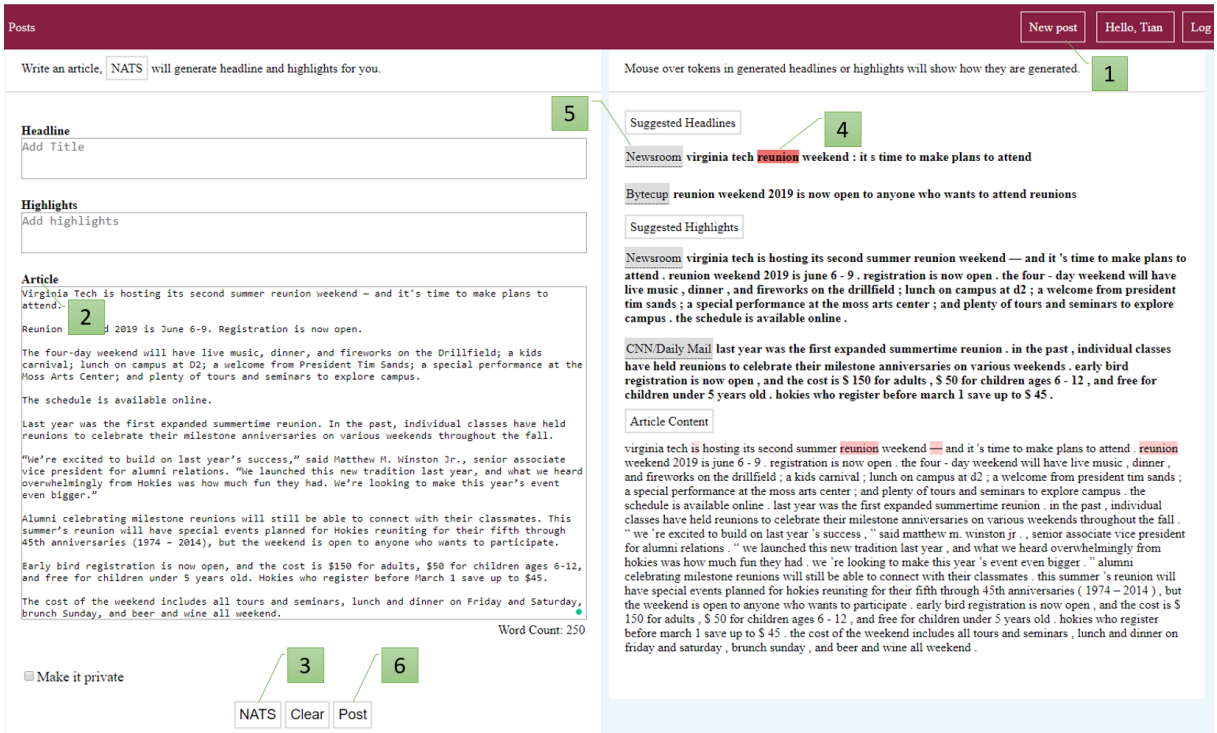

Figure 3: Front-end design of the live demonstration of our system.

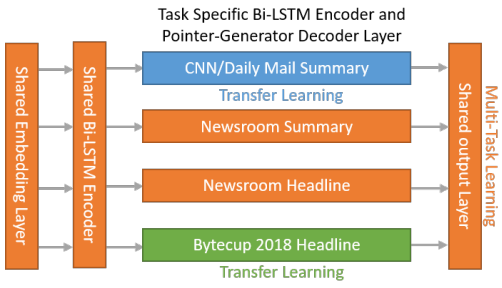

Figure 4: Overview of the model used to generate headlines and summaries.

ules provided in LeafNATS toolkit to assemble a new model (see Fig. 4), which has a shared embedding layer, a shared encoder layer, a task specific encoder-decoder (Bi-LSTM encoder and pointer-generator decoder) layer and a shared output layer.

To train this model, we first build a multi-task learning pipeline for Newsroom dataset to learn parameters for the modules that are colored in orange in Fig. 4, because (1) articles in this dataset have both headlines and highlights, (2) the size of the dataset is large, and (3) the articles come from a variety of news agents. Then, we build a transfer learning pipeline for CNN/Daily and Bytecup dataset, and learn the parameters for modules labeled with blue and green color, respectively. With LeafNATS, we can accomplish this work efficiently.

The performance of the proposed model on the corresponding testing sets are shown in Table 3. From the table, we observe that our model performs better in headline generation tasks. However, the ROUGE scores in summarization tasks are lower than the models without sharing embedding, encoder and output layers. It should be noted that by sharing the parameters, this model requires less than 20 million parameters to achieve such performance.

Dataset	Model	R-1	R-2	R-L
Newsroom-S	multi-task	39.85	28.37	36.91
Newsroom-H	multi-task	28.31	13.40	26.64
CNN/DM	transfer	35.55	15.19	33.00
	+coverage	38.49	16.78	35.68
Bytecup	transfer	40.92	24.51	38.01

Table 3: Performance of our model.

4 Conclusion

In this paper, we have introduced a LeafNATS toolkit for building, training, testing/evaluating, and deploying NATS models, as well as a live news blogging system to demonstrate how the NATS models can make the work of writing headlines and summaries for news articles more efficient. An extensive set of experiments on different benchmark datasets has demonstrated the effectiveness of our implementations. The newly proposed model for this system has achieved competitive results with fewer number of parameters.

Acknowledgments

This work was supported in part by the US National Science Foundation grants IIS-1619028, IIS-1707498 and IIS-1838730.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Asli Celikyilmaz, Antoine Bosselut, Xiaodong He, and Yejin Choi. 2018. Deep communicating agents for abstractive summarization. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 1662–1675.

Mahak Gambhir and Vishal Gupta. 2017. Recent automatic text summarization techniques: a survey. *Artificial Intelligence Review*, 47(1):1–66.

Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. 2018. Bottom-up abstractive summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4098–4109.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Max Grusky, Mor Naaman, and Yoav Artzi. 2018. Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 708–719.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701.

Zhiting Hu, Haoran Shi, Zichao Yang, Bowen Tan, Tiancheng Zhao, Junxian He, Wentao Wang, Xingjiang Yu, Lianhui Qin, Di Wang, et al. 2018. Texar: A modularized, versatile, and extensible toolkit for text generation. *arXiv preprint arXiv:1809.00794*.

Yaser Keneshloo, Tian Shi, Chandan K Reddy, and Naren Ramakrishnan. 2018. Deep reinforcement learning for sequence to sequence models. *arXiv preprint arXiv:1805.09461*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. *Proceedings of ACL 2017, System Demonstrations*, pages 67–72.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. *Text Summarization Branches Out*.

Thang Luong, Hieu Pham, and Christopher D Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421.

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.

Alexander Miller, Will Feng, Dhruv Batra, Antoine Bordes, Adam Fisch, Jiasen Lu, Devi Parikh, and Jason Weston. 2017. Parlai: A dialog research software platform. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 79–84.

Ramesh Nallapati, Feifei Zhai, and Bowen Zhou. 2017. Summarunner: A recurrent neural network based sequence model for extractive summarization of documents. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Ça glar Gulçehre, and Bing Xiang. 2016. Abstractive text summarization using sequence-to-sequence RNNs and beyond. *CoNLL 2016*, page 280.

Graham Neubig, Matthias Sperber, Xinyi Wang, Matthieu Felix, Austin Matthews, Sarguna Padmanabhan, Ye Qi, Devendra Singh Sachan, Philip Arthur, Pierre Godard, et al. 2018. Xnmt: The extensible neural machine translation toolkit. *Vol. 1: MT Researchers Track*, page 185.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch.

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

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

Alexander M Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083. Association for Computational Linguistics.

Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C Courville, and Joelle Pineau. 2016. Building end-to-end dialogue systems using generative hierarchical neural network models.

Tian Shi, Yaser Keneshloo, Naren Ramakrishnan, and Chandan K Reddy. 2018. Neural abstractive text summarization with sequence-to-sequence models. *arXiv preprint arXiv:1812.02303*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Rakesh M. Verma and Daniel Lee. 2017. Extractive summarization: Limits, compression, generalized model and heuristics. *Computación y Sistemas*, 21.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.