DeepZensols: A Deep Learning Natural Language Processing Framework for Experimentation and Reproducibility

Paul Landes, Barbara Di Eugenio, Cornelia Caragea
Department of Computer Science
University of Illinois at Chicago
{plande2, bdieugen, cornelia}@uic.edu

Abstract

Given the criticality and difficulty of reproducing machine learning experiments, there have been significant efforts in reducing the variance of these results. The ability to consistently reproduce results effectively strengthens the underlying hypothesis of the work and should be regarded as important as the novel aspect of the research itself. The contribution of this work is an open source framework that has the following characteristics: a) facilitates reproducing consistent results, b) allows hot-swapping features and embeddings without further processing and re-vectorizing the dataset, c) provides a means of easily creating, training and evaluating natural language processing deep learning models with little to no code changes, and d) is freely available to the community.

1 Introduction

Consistently reproducing results is a fundamental criterion of the scientific method, without which, a hypothesis may be weakened or even invalidated (Arvan et al., 2022). Reproduction of results becomes even more necessary as a growing number of publications are inflated by false positives (Head et al., 2015). Efforts to abate this trend include introducing new statistical methods to detect false findings (Ulrich and Miller, 2015).

The inability to reproduce results has been referred to as the "replication crisis" (Hutson, 2018). The problem of reproducibility in results is becoming more acknowledged as a serious issue in the machine learning (ML) community with efforts to understand and overcome the challenge (Rogers et al., 2021; Drummond, 2018). Not only has the community addressed the issue in the literature, it has endeavored to assess if experiments are reproducible and provide recommendations to enhance reproducibility as with the Reproducibility Challenge[1]. To address these issues, we present DeepZensols, a freely available[2] deep learning (DL) framework for NLP research by and for the academic research community including citizen scientists, academic researchers, and students. It has been used for research projects(Landes et al., 2022, 2023) funded by the National Institute of Health (NIH)[3].

A key feature that sets DeepZensols apart from others is a novel method to rapidly and easily swap features sets and compare performance across models (see Section 2.4). Other systems must re-parse and re-vectorize each mini-batch over each epoch. While there exist similar frameworks to ours (Ning et al., 2020; Falcon, 2019; Paszke et al., 2019; Alberti et al., 2018), none of these provides this batch strategy, vectorization of natural language text features and reproducibility of results across advanced programming interfaces (APIs) and datasets in one framework. Popular neural network (NN) architectures are available out of the box and easily configurable with little to no coding necessary (see Section 2.2 for NLP specific framework details).

2 Library Design

DeepZensols is a combination of Python APIs built on top of PyTorch that provide a means of easily and quickly creating NLP task specific pipelines. The framework's source code and installable libraries are released under the MIT Open Source License, and includes extensive and in depth overview and API documentation, tutorials, Jupyter Notebook examples and class diagrams for NLP reference models and datasets. The framework is validated with 381 unit tests and six integration tests, which are automated using continuous integration.

2.1 Reproducibility

All random state, including utility libraries, scientific libraries, PyTorch, and GPU state, is consistent

[1] https://www.cs.mcgill.ca/.../ReproducibilityChallenge.html

[2] https://github.com/plandes/deepnlp

[3] NIH award R01CA225446, MyPHA: Automatically generating personalized accounts of in-patient hospitalizations.

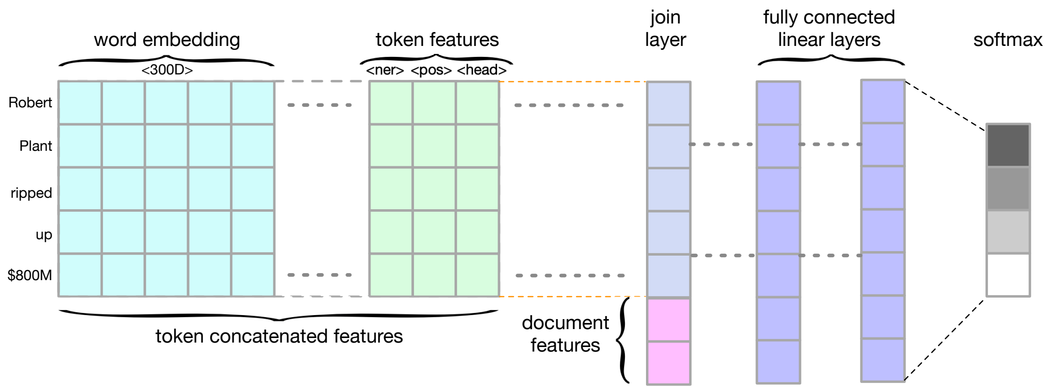

Figure 1: Word embeddings concatenated to vectorized linguistic features, and then joined with vectorized document features constructed using configuration with no coding.

across each run of the interpreter execution of the model's training, evaluation and testing when using the framework. Results are consistent by saving this random state when saving the model, then retrieving and resetting it before using the model.

The order of mini-batches, and their constituent data can affect the model performance as an aspect of training or the results of validation and testing (Pham et al., 2020). This performance inconsistency is addressed by recording the order of all data[4] and tracking the training, validation and test data splits. Not only are mini-batches given in the same order, the ordering in each mini-batch is also preserved. These dataset partitions and their order are saved to the file system so the community has the option of distributing it along with the source code for later experiment duplication.

The framework also saves the configuration used to recreate the same in-memory state along with the model. This duplicates all train-time memory model structures, parameters, and hyperparameters during testing. For the framework's reproducibility, unit tests are executed for individual components and integration tests by comparing the validation and training loss across six data sets[5]. In addition, this demonstrates to users of the framework how to add their own components and tests.

2.2 NLP-Focused Abstractions and Features

The framework provides many APIs for natural language tasks, including concatenation of vectorized language features to input embedding (see Figure 1). Vectorization of contextual embeddings such as BERT (Devlin et al., 2019) and non-contextual embeddings such as word2vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014) and fastText (Bojanowski et al., 2017) are available.

The framework includes many layer implementations, which are compatible with the PyTorch API as module classes. Examples of layers provided include BiLSTM CRF, BERT transformer models, 1D convolution NN, word embedding layer for concatenating features (see Section 2.3), and TF/IDF frequency weighting (Sparck Jones, 1972).

HuggingFace transformer layers are available as embeddings, document, sentence and token features. The framework also provides direct access to these models' data and utilizes it in a variety of tasks such as text classification, token classification, language generation, latent semantic analysis, etc. A linguistic feature mapper that translates spaCy[6] to wordpieces, which are token sub-units with associated vectors (Wu et al., 2016), is also accessible as an easy to configure module.

2.3 Vectorization

The DeepZensols framework allows for easily configurable components that provide a higher level abstraction that tokenizes, sentence chunks, and vectorizes linguistic features. These *vectorizers* have a class taxonomy based on data they vectorize so their output data can be automatically constructed in various off-the-shelf architectures. See Section 2.2 for more information on NLP specific feature generation.

2.4 Batching

We provide a novel method to vectorize and batch data without wasteful pre-processing of feature and

[4] Regardless of any given data pre-processing or shuffling.

[5] Data sets include the MNIST, Adult, Iris datasets and those mentioned in Section 3.

[6] https://spacy.io

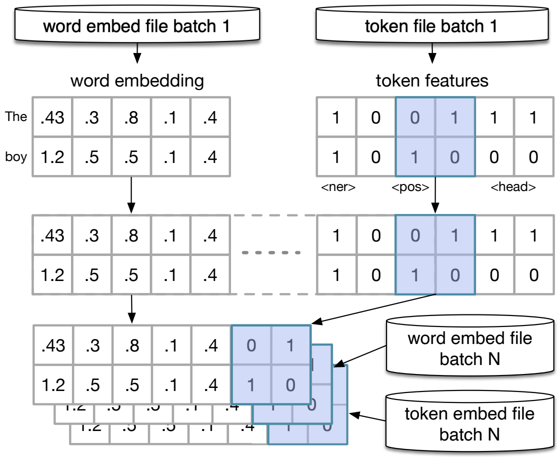

Figure 2: Batch decoding "stitches" mini-batches together from files containing features for the current run.

embedding combinations. Other similar frameworks pre-process data in an intermediate form only once before training. However, this leads to a brittle and difficult to reproduce dataset of ad-hoc text processing scripts that are challenging to re-execute, and thus, reproduce performance metrics.

Our framework addresses this with an organized intermediate file scheme and partitioned feature set so the input data is vectorized only once efficiently using a multi-processing pipeline. The output format of this process allows for quick feature swapping and hyperparameter tuning for re-training. It leverages the fact that mini-batches are independent and fit nicely as independent units of work by segmenting datasets into smaller chunks, vectorizing each chunk in parallel sub-processes, and creating batches independently across each sub processes.

This process by which data is written to the file system in a format that is fast to reassemble is called *batch encoding* and accomplished by: a) split sentences and/or tokens into equal size "chunks" units of work, b) parsing natural language features from chunks across multiple processes, and c) vectorizing each chunk as tensor data in separate files by feature.

After batch encoding is complete, the model is ready to be trained from data obtained from a *batch decoding* step, which is accomplished by: a) choosing a feature set for a training run, b) reassembling features by mini-batch, c) decode each mini-batch into a tensor (see Figure 2), and d) load, cache and copy tensors to the GPU.

Reassembling mini-batches by feature greatly reduces load time and memory space, which speeds up model training (see Section 3) and ameliorates issues of complex models. The train, validation and test cycle is faster for other vectorized linguistic data such as spaCy features as well.

2.5 Execution

The framework provides both a command line and a Jupyter notebook interface to train, test and predict. A "glue" API is used to make a Python dataclass[7] class a dynamically generated command line with help usage message documentation. A set of default application classes are available with the framework, but they can be extended to include project specific workflows. The default application set provides interactive early stopping or epoch resetting during training.

Results are organized by each run and carry a common file system structured named by either what is provided in the configuration or by model name. This directory structure contains the full model with all configuration, the PyTorch model, and results provided as human readable indented text, JSON and binary formats.

3 Runtime Analysis

Runtime analysis was performed for parsing, feature vectorization (see Section 2.3), batching (see Section 2.4), training and testing three different types of models using a Nvidia TITAN RTX graphics processor on an Intel 3.6GHz CPU using the following criteria:

- Model: the model trained and evaluated.
- Batch: whether or not the mini-batches were (re)created (see Section 2.4).
- GPU: whether or not the mini-batches were cached in GPU memory.[8]

Since obtaining fast results allows for more experimentation with a variety of feature sets, embeddings, and NN architectures, our experiments included several combinations of caching strategies. Table 1 shows the latency to batch, retrain and test the model for each dataset in the "Duration" column. Experiments were rerun obtain the time needed for training, validation and testing of each model, then a second time using the precomputed mini-batched data. The GPU caching option was toggled across these experiments to find the CPU to GPU latency for loading mini-batches.

[7] https://docs.python.org/3/library/dataclasses.html

[8] The framework offers GPU caching, CPU caching, and iterative buffering of mini-batches.

Data	Model	Duration	Batch	GPU	Both
NER	BERT	1:06:04	04:23	00:12	04:35
	GloVe	34:08	04:19	05:41	10:00
Mov	BERT	21:19	02:04	-00:26	01:38
	GloVe	05:03	03:07	01:20	04:27
CB	BERT	05:48	01:50	-00:01	01:49
	GloVe	05:45	01:51	03:03	04:54

Table 1: Efficiency benchmarks showing the **N**amed **E**ntity **R**ecognition, **Mov**ie review sentiment, and **C**lick**B**ate datasets. The "Duration" column lists processing latency with no batch or GPU caching in hours, minutes and seconds. The "Batch" and "GPU" columns have the caching speedup times in minutes and seconds. The "Both" column is the speedup with both batch and GPU caching are enabled.

The datasets used in the runtime analysis include the CoNNL 2003 (Tjong Kim Sang and De Meulder, 2003) for NER, the movie review corpus (Pang and Lee, 2005; Socher et al., 2013) for sentiment, and the clickbate corpus (Chakraborty et al., 2016) for text classification.

The results show significant processing improvements in all three datasets with the GloVe model leading. This is likely due to how the static embedding are not computed for each sentence (unlike BERT). The NER dataset with the BERT model was faster by 4.5 minutes and the GloVe model was 10 minutes faster (1.4X speedup). However, the movie review sentiment dataset shows the best improvement (7.8X speedup) on the GloVe model. This is primarily from the batch caching 2.6X speedup, but benefited from a GPU 1.3X caching speedup. We hypothesize that the GPU slowdowns for the movie review and clickbate datasets are potentially due to larger BERT (768D vs 300D embeddings) mini-batches copied from the CPU.

4 Related Frameworks

Popular DL frameworks such as TensorFlow[9] have a dashboard that provides metrics, such as training and validation loss. However, these general purpose frameworks offer basic performance metrics and do not provide a means of producing higher abstraction level NLP specific models. More specifically, frameworks such as Keras, supply a very coarse API allowing solely for cookie-cutter models. They lack the ability to easily create and evaluate models past this surface interface.

Frameworks such as PyTorch[10], which are more common in academia, provide a more straightforward simple API that is similar to the core TensorFlow libraries, and thus have the same shortcomings as a tool to bridge the gap between pure research and reproducibility. Specifically, they do not provide batching for accessible feature swapping and ablation studies, or retention of ML algorithm state necessary to reproduce results.

AllenNLP (Gardner et al., 2018) is a flexible configuration driven framework that provides construction of NLP NN architectures and is the closest framework to ours. However, it does not have fast feature swapping (see Section 2.4) and batch creation capability, and lacks most of the components necessary to consistently reproduce results[11].

Popular packages providing support for transformer architectures such as BERT (Devlin et al., 2019) include HuggingFace[12]. However, this framework only provides transformer models for contextual word embeddings.

5 Conclusion and Limitations

The DeepZensols framework is a viable solution to easily create NLP specific models with APIs and analysis tools to produce consistent results. Such frameworks create the types of models that give confidence and legitimacy by providing a way to produce reliable reproducible results for researchers not familiar with deep learning tools, practitioners, medical personnel, students, and those new to the field. Runtime analysis shows the framework offers significant processing time savings compared to systems that do not provide feature caching with stable results, but not all HuggingFace pretrained models[13] have been tested. The following have been tested: BERT, RoBERTa (Liu et al., 2019), DistilBERT (Sanh et al., 2019), Big Bird (Zaheer et al., 2020), BioBERT (Lee et al., 2020), XML-R (Conneau et al., 2020), ClinicalBioBERT (Alsentzer et al., 2019), and GatorTron (Yang et al., 2022) have been tested. A planned future work is to integrate the framework with TensorBoard[14].

6 Acknowledgments

This work is partially supported by award R01CA225446 from the NIH.

[9] https://www.tensorflow.org

[10] https://pytorch.org

[11] https://github.com/allenai/allennlp/issues/3100

[12] https://huggingface.co

[13] https://huggingface.co/models

[14] https://www.tensorflow.org/tensorboard

References

Michele Alberti, Vinaychandran Pondenkandath, Marcel Würsch, Rolf Ingold, and Marcus Liwicki. 2018. DeepDIVA: A Highly-Functional Python Framework for Reproducible Experiments. In *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 423–428.

Emily Alsentzer, John Murphy, William Boag, Wei-Hung Weng, Di Jindi, Tristan Naumann, and Matthew McDermott. 2019. Publicly Available Clinical BERT Embeddings. In *Proceedings of the 2nd Clinical Natural Language Processing Workshop*, pages 72–78. Association for Computational Linguistics.

Mohammad Arvan, Luís Pina, and Natalie Parde. 2022. Reproducibility in Computational Linguistics: Is Source Code Enough? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2350–2361. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Abhijnan Chakraborty, Bhargavi Paranjape, Sourya Kakarla, and Niloy Ganguly. 2016. Stop clickbait: Detecting and preventing clickbaits in online news media. In *Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ASONAM '16, pages 9–16. IEEE Press.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised Cross-lingual Representation Learning at Scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186.

Chris Drummond. 2018. Reproducible research: A minority opinion. *Journal of Experimental & Theoretical Artificial Intelligence*, 30(1):1–11.

William A Falcon. 2019. Pytorch lightning. *GitHub*, 3.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew E. Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A Deep Semantic Natural Language Processing Platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6.

Megan L. Head, Luke Holman, Rob Lanfear, Andrew T. Kahn, and Michael D. Jennions. 2015. The Extent and Consequences of P-Hacking in Science. *PLoS Biology*, 13(3).

Matthew Hutson. 2018. Artificial intelligence faces reproducibility crisis. *Science*, 359(6377):725–726.

Paul Landes, Aaron Chaise, Kunal Patel, Sean Huang, and Barbara Di Eugenio. 2023. Hospital Discharge Summarization Data Provenance. In *The 22nd Workshop on Biomedical Natural Language Processing and BioNLP Shared Tasks*, pages 439–448. Association for Computational Linguistics.

Paul Landes, Kunal Patel, Sean S. Huang, Adam Webb, Barbara Di Eugenio, and Cornelia Caragea. 2022. A New Public Corpus for Clinical Section Identification: MedSecId. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 3709–3721. International Committee on Computational Linguistics.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. 2020. BioBERT: A pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv: 1907.11692 (Only available as arXiv preprint).

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*, volume 26. Curran Associates, Inc.

Qiang Ning, Hao Wu, Pradeep Dasigi, Dheeru Dua, Matt Gardner, Robert L. Logan Iv, Ana Marasović, and Zhen Nie. 2020. Easy, Reproducible and Quality-Controlled Data Collection with CROWDAQ. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 127–134.

Bo Pang and Lillian Lee. 2005. Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 115–124. Association for Computational Linguistics.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward

Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

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. Association for Computational Linguistics.

Hung Viet Pham, Shangshu Qian, Jiannan Wang, Thibaud Lutellier, Jonathan Rosenthal, Lin Tan, Yaoliang Yu, and Nachiappan Nagappan. 2020. Problems and opportunities in training deep learning software systems: An analysis of variance. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 771–783. Association for Computing Machinery.

Anna Rogers, Timothy Baldwin, and Kobi Leins. 2021. 'Just What do You Think You're Doing, Dave?' A Checklist for Responsible Data Use in NLP. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 4821–4833. Association for Computational Linguistics.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. In *The 5th EMC2 - Energy Efficient Training and Inference of Transformer Based Models*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642. Association for Computational Linguistics.

Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Rolf Ulrich and Jeff Miller. 2015. P-hacking by post hoc selection with multiple opportunities: Detectability by skewness test?: Comment on Simonsohn, Nelson, and Simmons (2014). *Journal of Experimental Psychology: General*, 144(6):1137–1145.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. arXiv: 1609.08144 (Only available as arXiv preprint).

Xi Yang, Aokun Chen, Nima PourNejatian, Hoo Chang Shin, Kaleb E. Smith, Christopher Parisien, Colin Compas, Cheryl Martin, Mona G. Flores, Ying Zhang, Tanja Magoc, Christopher A. Harle, Gloria Lipori, Duane A. Mitchell, William R. Hogan, Elizabeth A. Shenkman, Jiang Bian, and Yonghui Wu. 2022. GatorTron: A Large Clinical Language Model to Unlock Patient Information from Unstructured Electronic Health Records. arXiv: 2203.03540 (Only available as arXiv preprint).

Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontañón, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big Bird: Transformers for Longer Sequences. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.