AutoAttend: Automated Attention Representation Search

Chaoyu Guan [1] **Xin Wang** [1] **Wenwu Zhu** [1]

A. Datasets and Baselines

This section describes in detail the datasets we use and baselines we compare.

A.1. Natural Language Processing

A.1.1. Datasets

Standford Sentiment Treebank (SST) This is a dataset published by Standford NLP library for sentiment analysis. All sentences are movie reviews from rottentomatoes.com together with its sentence structure and fine-grained sentiment label on each sub-structure. However, in this paper, we do not use sentence structure in our model to stay consistent with other datasets we use. There are two version of SST dataset: full version and binary version. The full version includes 5 level sentiments, while the binary verion has only positive and negative sentiments. See https://nlp.stanford.edu/sentiment/index.html for the detailed information.

AG News (AG) AG news corpus are collected by a news dataset collected by an academic news search engine *ComeToMyHead*. The corpus website is http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html. Then, Zhang et al. (2015) construct a four-class news topic classification dataset from it for the use of text classification by choosing the four largest topics in the original dataset. The dataset can be found at https://github.com/mhjabreel/CharCnn_Keras/tree/master/data/ag_news_csv.

DBPedia Ontology (DBP) This dataset is collected by Zhang et al. (2015) from the structural database *DBPedia* constructed on Wikipedia, among which, 14 non-overlapping ontology classes are chosen to form the document classification DBPedia dataset.

Yelp Reviews (YELP) This dataset is collected by Zhang et al. (2015) from Yelp Dataset Challenge in 2015. The full version aims at predicting the origin stars the user has given. The binary version (YELP-B) is constructed by setting 1,2 as negative and 3,4 as positive.

Yahoo! Answers (YAHOO) This dataset is collected by Zhang et al. (2015) from Yahoo! Answers Comprehensive Questions and Answers version 1.0 dataset. The data from the top-10 largest topics are selected to construct the Yahoo! Answers dataset.

Amazon Reviews Binary (AMZ-B) This dataset is the binary version of Amazon Reviews dataset in Zhang et al. (2015), which is collected from Stanford Network Analysis Project, and is the largest text classification dataset used in this paper.

In this paper, the SST dataset is obtained and proposed through the same procedure in TextNAS (Wang et al., 2020)[1]. The max length is set to 64 and all the sentence spans are used to train the model. For the rest of datasets we use, we utilize torchtext[2] to obtain the datasets, and manually keep a vocabulary 50000 and set the rest of words to _unk_ token.

A.1.2. Baselines

For text classification baselines, we only compare with those who are 1) word-level model 2) do not rely on structure information and 3) do not use extra external data except GloVe (Pennington et al., 2014) for a fair comparison.

For hand-crafted baselines, we select current state-of-the-art hand-crafted baselines Gumbel-LSTM (Choi et al., 2018), CAS-LSTM (Choi et al., 2019) for SST dataset, and DNC (Le et al., 2019), DAGRN (Liu et al., 2020), Global-local encoders (Niu et al., 2019) for the rest of datasets. We report their scores borrowing from the corresponding papers, and leave the unreported scores as - in the Table 4.

For searched baselines, we compare our AutoAttend framework with previous state-of-the-art NAS models TextNAS (Wang et al., 2020), where they construct a search space similar to our baseline in Section 4.1.1 and leverage intra-layer self-attention as a primitive operation. Various baseline

[1]Tsinghua University. Correspondence to: Xin Wang <xin_wang@tsinghua.edu.cn>, Wenwu Zhu <wwzhu@tsinghua.edu.cn>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

[1]https://github.com/microsoft/nni/tree/master/examples/nas/textnas

[2]https://pytorch.org/text/stable/datasets.html

search algorithms are also performed in the TextNAS search space, including DARTS (Liu et al., 2019), SMASH (Brock et al., 2018), One-Shot (Bender et al., 2018), and random search (Li & Talwalkar, 2019). The performance scores are borrowed directly from TextNAS (Wang et al., 2020).

A.2. Graph Representation Learning

A.2.1. Datasets

Citation Network Datasets The transductive datasets we use are collected and splitted by Yang et al. (2016). There are three datasets in total, namely **Cora**, **Citeseer**, and **Pubmed**. Each node represents a document, and their bag-of-words features are collected as their node features. When two documents have citation relationship, the two nodes in graph are connected together. The tasks are classifying the documents on each nodes.

Protein-protein interaction networks (PPI) The inductive dataset we use are collected by ZitNik et al. (2017). The PPI dataset contains multiple graphs that represent different human tissues. The Gene Ontology are used as labels for each nodes for classification.

For the use of datasets above, we directly those provided by PyTorch Geometric[3]. The data split is used following the original settings in the corresponding datasets and is the same for all baselines and our works.

A.2.2. Baselines

For hand-crafted baselines, we select popular and state-of-the-art models GCN (Kipf & Welling, 2017), (Velickovic et al., 2018), arma (Bianchi et al., 2019), and appnp (Klicpera et al., 2019). All the scores of baselines are reruned by ourselves using implementations provided by PyTorch Geometric[4].

For searched baselines, we compare our AutoAttend to current state-of-the-art NAS baselines GraphNAS (Gao et al., 2020) and AGNN (Zhou et al., 2019). We notice that for the codebase[5] given officially by GraphNAS, there exists test data leakage in their implementation of re-train. Thus, we fix the leakage and rerun their re-train codes directly, and report the revised scores in Table 5. For AGNN, there are no publicly available codes. We directly adopt the scores they report in their paper to the Table 5.

[3] https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html

[4] https://github.com/rusty1s/pytorch_geometric/tree/master/examples

[5] https://github.com/GraphNAS/GraphNAS

B. Primitive Operation Pool

This section describes in detail the primitive operation pool we use for NLP and GRL.

B.1. Natural Language Processing

Conv Convolution layers borrowed from Convolutional Neural Network (LeCun et al., 1989). We only perform convolution on sequence direction (known as 1-D Convolution[6]), with Layer Normalization (Ba et al., 2016) and `ReLU` activations applied before and after convolutions. In this paper, we adopt Conv1 and Conv3 as selectable operations.

Max Pooling Similar to convolution layers without the activation functions, but replace the convolutional kernel with a max-pooling kernel, which only select the max value as outputs when "convolution" on given receptive fields (known as 1-D MaxPooling[7]). The kernel size is kept as 3 in our search space.

Gated Recurrent Unit (GRU) (Cho et al., 2014) A variance of LSTM, but with fewer gates, which leads to higher computation efficiency while mantaining the same effectiveness. We adopt a bi-directional form of GRU. Same as Conv and Max Pooling, we also add Layer Normalization before GRU cell.

Note that we do not include more complicated operations like Conv5, Conv7, Mean-Pooling, and Intra-layer Self-Attention in TextNAS (Wang et al., 2020), since we find in our priliminary experiments that, these operations are redundant when searched under the space settings of AutoAttend.

B.2. Graph Representation Learning

The description of operations are already detailed in the Section 5.2.1 and we only add some complementary information here. For dataset Cora and PPI, we find that the space settings described in Section 5.2.1 are not very suitable. Therefore, we borrow the primitive operation pool of micro search space given in GraphNAS (Gao et al., 2020) as building blocks directly in our macro search space.

Specifically, we use the following operations as our macro search space primitive operation pool:

GCN The Graph Convolutional Network layer borrowed from Kipf & Welling (2017).

SAGE-MEAN The Graph Sage network layer borrowed from Hamilton et al. (2017).

[6] https://pytorch.org/docs/stable/generated/torch.nn.Conv1d.html?highlight=conv1d#torch.nn.Conv1d

[7] https://pytorch.org/docs/stable/generated/torch.nn.MaxPool1d.html?highlight=maxpool1d#torch.nn.MaxPool1d

Table A1. **AutoAttend** text classification transfer setting

DATASET	BATCH SIZE	MAX LENGTH	LR	WINDOW SIZE	STRIDE	HIDDEN SIZE	DROPOUT
AG	128	256	0.0005	-	-	256	0.1
DBP	128	256	0.0005	-	-	256	0.1
YELP-B	128	512	0.0005	-	-	64	0.1
YELP	128	512	0.0005	-	-	64	0.1
YAHOO	128	1024	0.0005	64	32	32	0.1
AMZ-B	128	256	0.0005	-	-	128	0.1

GAT The Graph Attention Network layer borrowed from Velickovic et al. (2018). We choose head number 16, 8, 4, 2, 1 to give five different primitive operations.

Linear Linear tranform layer performed on node features.

ARMA The ARMA graph convolutional layer borrowed from Bianchi et al. (Bianchi et al., 2019).

CHEB The chebyshev spectral graph convolutional layer borrowed from Defferrard et al. (2016).

SGC The simple graph convolutional layer borrowed from Wu et al. (2019).

For the implementations of all layers above, we directly use the one provided by PyTorch Geometric[8].

C. Transfer Settings

For a fair comparison, we basically follow the settings provided by TextNAS (Wang et al., 2020) to transfer our text encoder architecture on other text classification datasets. For all datasets, we only keep 50000 max vocabularies to reduce the space complexity. We also use slide window tricks to solve optimization problems in long sentences: a window of fixed size is slided over sentences with fixed strides, with each window outputting a sentence vector through our sentence encoder. Then, global max pooling is performed to derive the final sentence representation. All the detailed transfer information are given in Table A1.

References

Ba, L. J., Kiros, J. R., and Hinton, G. E. Layer normalization. *CoRR*, abs/1607.06450, 2016.

Bender, G., Kindermans, P., Zoph, B., Vasudevan, V., and Le, Q. V. Understanding and simplifying one-shot architecture search. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pp. 549–558, 2018.

Bianchi, F. M., Grattarola, D., Livi, L., and Alippi, C. Graph neural networks with convolutional ARMA filters. *CoRR*, abs/1901.01343, 2019.

Brock, A., Lim, T., Ritchie, J. M., and Weston, N. SMASH: one-shot model architecture search through hypernetworks. In *ICLR*, 2018.

Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. In *SSST@EMNLP*, pp. 103–111, 2014.

Choi, J., Yoo, K. M., and Lee, S. Learning to compose task-specific tree structures. In *AAAI*, pp. 5094–5101, 2018.

Choi, J., Kim, T., and Lee, S. Cell-aware stacked lstms for modeling sentences. In *ACML*, volume 101 of *Proceedings of Machine Learning Research*, pp. 1172–1187, 2019.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *NeurIPS*, pp. 3837–3845, 2016.

Gao, Y., Yang, H., Zhang, P., Zhou, C., and Hu, Y. Graph neural architecture search. In *IJCAI*, pp. 1403–1409, 2020.

Hamilton, W. L., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *NeurIPS*, pp. 1024–1034, 2017.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *ICLR*. OpenReview.net, 2017.

Klicpera, J., Bojchevski, A., and Günnemann, S. Predict then propagate: Graph neural networks meet personalized pagerank. In *ICLR*. OpenReview.net, 2019.

Le, H., Tran, T., and Venkatesh, S. Learning to remember more with less memorization. In *ICLR*, 2019.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, 1989.

[8] https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html

Li, L. and Talwalkar, A. Random search and reproducibility for neural architecture search. In *UAI*, pp. 367–377, 2019.

Liu, H., Simonyan, K., and Yang, Y. DARTS: differentiable architecture search. In *ICLR*. OpenReview.net, 2019.

Liu, Y., Meng, F., Chen, Y., Xu, J., and Zhou, J. Depth-adaptive graph recurrent network for text classification. *CoRR*, abs/2003.00166, 2020.

Niu, G., Xu, H., He, B., Xiao, X., Wu, H., and Gao, S. Enhancing local feature extraction with global representation for neural text classification. In *EMNLP*, pp. 496–506, 2019.

Pennington, J., Socher, R., and Manning, C. D. Glove: Global vectors for word representation. In *EMNLP*, pp. 1532–1543, 2014.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *ICLR*. OpenReview.net, 2018.

Wang, Y., Yang, Y., Chen, Y., Bai, J., Zhang, C., Su, G., Kou, X., Tong, Y., Yang, M., and Zhou, L. Textnas: A neural architecture search space tailored for text representation. In *AAAI*, pp. 9242–9249, 2020.

Wu, F., Jr., A. H. S., Zhang, T., Fifty, C., Yu, T., and Weinberger, K. Q. Simplifying graph convolutional networks. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6861–6871. PMLR, 2019.

Yang, Z., Cohen, W. W., and Salakhutdinov, R. Revisiting semi-supervised learning with graph embeddings. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pp. 40–48. JMLR.org, 2016.

Zhang, X., Zhao, J. J., and LeCun, Y. Character-level convolutional networks for text classification. In *NeurIPS*, pp. 649–657, 2015.

Zhou, K., Song, Q., Huang, X., and Hu, X. Auto-gnn: Neural architecture search of graph neural networks. *CoRR*, abs/1909.03184, 2019.

Zitnik, M. and Leskovec, J. Predicting multicellular function through multi-layer tissue networks. *Bioinform.*, 33(14): i190–i198, 2017.