TextNAS: A Neural Architecture Search Space Tailored for Text Representation

Yujing Wang,[1,2,*] **Yaming Yang,**[1,*] **Yiren Chen,**[2,*,†] **Jing Bai,**[1] **Ce Zhang**[3,†]

Guinan Su,[4,†] **Xiaoyu Kou,**[2,†] **Yunhai Tong,**[2] **Mao Yang,**[1] **Lidong Zhou**[1]

[1]Microsoft Research Asia

[2]Key Laboratory of Machine Perception, MOE, School of EECS, Peking University

[3]ETH Zürich, [4]University of Science and Technology of China

{yujwang, yayaming, jbai, maoyang, lidongz}@microsoft.com

{yrchen92, kouxiaoyu, yhtong}@pku.edu.cn

ce.zhang@inf.ethz.ch, sa517299@mail.ustc.edu.cn

Abstract

Learning text representation is crucial for text classification and other language related tasks. There are a diverse set of text representation networks in the literature, and how to find the optimal one is a non-trivial problem. Recently, the emerging Neural Architecture Search (NAS) techniques have demonstrated good potential to solve the problem. Nevertheless, most of the existing works of NAS focus on the search algorithms and pay little attention to the search space. In this paper, we argue that the search space is also an important human prior to the success of NAS in different applications. Thus, we propose a novel search space tailored for text representation. Through automatic search, the discovered network architecture outperforms state-of-the-art models on various public datasets on text classification and natural language inference tasks. Furthermore, some of the design principles found in the automatic network agree well with human intuition.

Introduction

Neural network models have demonstrated their superiority in many natural language tasks such as text classification, machine translation and reading comprehension. One of the core problems of natural language processing is to design a network architecture that effectively captures the syntax and semantics incorporated in texts. Contrary to the computer vision domain where CNN is predominant, the state-of-the-art neural networks for text representation are much more diverse, including CNN (Zhang, Zhao, and LeCun 2015), RNN (Liu et al. 2015), hybrid model of CNN+RNN (Zhou et al. 2015; Tang, Qin, and Liu 2015) and Transformer (Vaswani et al. 2017), etc. Nevertheless, how to find the optimal text representation network is still an unsettled problem in the literature.

Recently, Neural Architecture Search (NAS) techniques have opened up a new opportunity for customized architecture design. Existing works of NAS mainly focus on the study of search algorithms and put little emphasis on the search space. However, there remain several challenges for applying NAS to different applications. First, it is prohibitive to search for all kinds of possibilities thoroughly, even when advanced search algorithms (for example, gradient-based, evolution, reinforcement learning, etc.) are utilized; Second, when the search space is extra-large, the NAS algorithm may select a neural architecture that overfits to both training and validation data. Thus, we argue that the search space is an indispensable human prior which deserves more investigation in different applications.

In this paper, we propose TextNAS, a novel search space customized for text representation. The search space is designed based on the following motivations and findings:

- **It is beneficial to explore the customized solution of layer mixture.** It is well-known that different layers are beneficial from different perspectives. CNN is good at learning local feature combinations (analogies to n-grams), RNN specializes in sequential modeling, and Transformer (Vaswani et al. 2017) is able to capture long-distance dependencies directly. There are some evidences demonstrating the potential of layer mixture, for instance, C-LSTM (Zhou et al. 2015) utilizes CNN to extract a sequence of higher-level phrase representation and then feeds the CNN output to another RNN layer to produce the ultimate sentence embedding vectors.

- **The macro search space is a better choice for text representation** Most previous works of NAS prefer micro search space (Zoph et al. 2017) as they work well on image-related tasks. However, according to a preliminary experiment (showed in Table 1), we demonstrate that the macro search space is better than the micro one in the text classification scenario. This shows the necessity of leveraging customized search spaces for different applications.

- **The search space should support multi-path ensembles.** One limitation of existing macro search space is that it only embodies single-path neural networks. However, multi-path ensemble is a common design principle in manual networks, e.g., InceptionV4 (Szegedy et al. 2017). Intuitively, different categories of layers act as distinct feature extractors, an ensemble of which provides potentially better representation for the sentence.

The TextNAS search space consists of a mixture of con-

Table 1: Comparison of micro and macro search spaces on different tasks using ENAS (Pham et al. 2018) search algorithm

Dataset	Task	Acc (micro)	Acc (macro)
CIFAR10	Image Classification	**97.11**	95.67
SST	Text Classification	47.00	**51.55**
YAHOO	Text Classification	70.63	**73.16**
AMZ	Text Classification	58.27	**62.64**

volutional, recurrent, pooling and self-attention layers. It is based on a general DAG structure and supports the ensemble of multiple paths. Given the search space, the TextNAS pipeline can be conducted in three procedures.[1] (1) The ENAS (Pham et al. 2018) search algorithm is performed on the search space by utilizing the evaluation accuracy on validation data as RL reward; (2) Grid search is conducted by the optimal architecture to search for the best hyper-parameter setting on the validation set. (3) The derived architecture is trained from scratch with the best hyper-parameters on the combination of training and validation data.

We ran experiments on the Stanford Sentiment Treebank (SST) dataset (Socher et al. 2013) to evaluate the TextNAS pipeline. The experimental results showed that the automatically generated neural architectures achieved superior performances compared to manually designed networks. We look into the automatic architecture and find that some of the design principles agree well with human experiences. Moreover, since the neural architecture search procedure is time- and resource-consuming, we are interested in the transferability of the derived network architectures to other text-related tasks. Impressively, the transferred architectures outperformed current state-of-the-art methods (Zhang, Zhao, and LeCun 2015; Yang et al. 2016; Conneau et al. 2016) on various text classification and natural language inference datasets.

Related Work

Neural Architecture Search

Neural Architecture Search (NAS) has become an important research topic in AutoML domain, the goal of which is to find the optimal network structure in a given search space which achieves excellent performance on a specific task. Existing studies in this direction can be summarized in two aspects. One line of research focuses on evolution algorithms, which offer flexible approaches for generating neural networks by simultaneously evolving along network structures and hyper-parameters (Real et al. 2018). Another line of research concentrates on reinforcement learning, for example, NAS (Neural Architecture Search) (Zoph and Le 2016) leverages a recurrent neural network as controller to generate child networks, while the controller is trained with reinforcement learning. Despite of impressive performance, the original NAS framework is computationally expensive.

There are various attempts to improve the search efficiency of NAS. (Zoph et al. 2017) reduces the search space to two

---

[1]The open source code is available at: https://github.com/yujwang/TextNAS

micro cells: the normal cell and the reduction cell, while the cells can be stacked to construct deep neural networks; PNAS (Liu et al. 2017) adopts a sequential model-based optimization strategy and constructs the network layer by layer while simultaneously learns a surrogate model to guide the search routine; (Baker et al. 2017) accelerates the search procedure through predicting the final performance by partially trained model configurations; ENAS (Pham et al. 2018) accelerates the reinforcement learning procedure by sharing parameters among child trials; DARTS (Liu, Simonyan, and Yang 2018) formulates the task of neural architecture search in a differentiable manner and does not require reinforcement learning controllers; SMASH (Brock et al. 2017) proposes one-shot model architecture search by designing a hyper-network to generate the parameter values for each model; (Bender et al. 2018) demonstrates the possibility of leveraging one-shot architecture search to identify promising architectures without hyper-networks or reinforcement learning; (Li and Talwalkar 2019) shows that random search with early-stop is a competitive NAS baseline and random search with weight-sharing achieves further improvement.

Text Classification

RNN is specialized for long sequential modeling and has the capability of processing variable-length inputs, making it a natural choice for text classification. For example, (Tai, Socher, and Manning 2015) introduces a tree-structured LSTM network to capture sentence meanings with emphasis on the syntactic structure. At the same time, there is another branch of methods using CNN for text classification (dos Santos and Gatti 2014; Zhang, Zhao, and LeCun 2015; Conneau et al. 2016). Benefit from the advantages of both RNN and CNN, there is a growing interest in assembling them, including C-LSTM (Zhou et al. 2015), RCNN (Kalchbrenner and Blunsom 2013) and GatedNN (Tang, Qin, and Liu 2015). These models utilize CNN to extract a sequence of higher-level phrase representation and feed the CNN output to additional RNN layers to produce the ultimate text representation vectors. Moreover, attention mechanism (Luong, Pham, and Manning 2015) has been widely adopted in NLP applications, which enables neural networks to focus on specific parts in the text sequence. As an example, (Yang et al. 2016) proposes a hierarchical attention network where two attention layers are applied at word and sentence level respectively. In addition, Transformer (Vaswani et al. 2017) invents multi-head self-attention in the text encoder to relate different positions of a single word sequence.

Natural Language Inference

Natural Language Inference (NLI) is another fundamental NLP task that determines the inferential relationship among sentences. There are two major categories of neural network models for NLI, namely sentence vector-based models and joint models. The former represents each sentence as a fixed-length vector before inferring the relationship between them; while the latter utilizes cross-sentence layers explicitly in the neural network for relation prediction. In this paper, the goal is to evaluate the capability of text representation, so we adopt
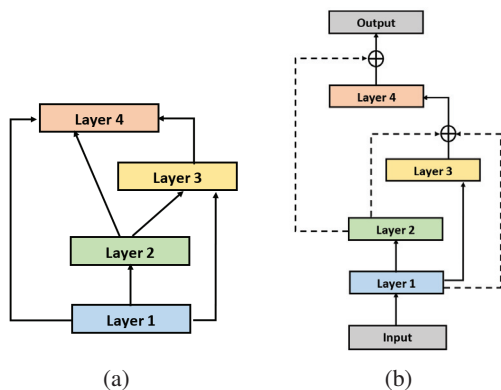
Figure 1: (a) The general DAG search space of four layers (b) A neural network instance sampled from the general search space.

the sentence-vector based framework. Conneau et al. (Conneau et al. 2017) compared 7 different network architectures and showed that a single BiLSTM layer with max pooling can act as the universal sentence encoding model. Based on this work, (Nie and Bansal 2017) designed a stacked BiLSTM layer with shortcut connections and (Talman, Yli-Jyrä, and Tiedemann 2018) devised a hierarchical BiLSTM max pooling (HBMP) model. Besides, (Chen, Ling, and Zhu 2018) proposed a new vector-based multi-head attention pooling layer to enhance the sentence representation; (Im and Cho 2017) utilized the self-attention network that considered local dependencies of different words to generate distance-based sentence embedding vectors; (Yoon, Lee, and Lee 2018) combined the self-attention mechanism with modified dynamic routing borrowed from the capsule network.

TextNAS

In this section, we introduce our method in details. First, we propose the novel search space tailored for text representation. Second, we introduce the search algorithms adopted in TextNAS. Finally, we describe the frameworks of two tasks, i.e., text classification and natural language inference.

Search Space

The macro search space of neural network can be depicted by a general DAG. As shown in Figure 1a, every node in the DAG represents a layer, and every edge from node i to node j denotes that layer i is served as an input or skip-connection to layer j. Without loss of generality, we define a topological order for the layers, where layer 0 stands for the original input layer and an edge $<i, j>$ exists when $i < j$. Based on the DAG search space, a network instance can be sampled by traversing the layers according to the topological order. For each layer i, we first choose a unique input layer from one of the previous layers $\{0, 1, ..., i-1\}$; then we make multiple choices from previous layers as skip connections, which are summed with the output of layer i. An example of the network instance is shown in Figure 1b, which can
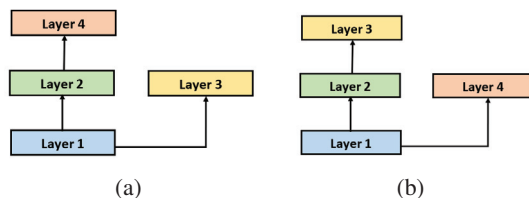

Figure 2: Duplicated network exmaples constructed by different orders

be generated in the following steps: (1) layer 2 and 3 both choose layer 1 as input; (2) layer 3 chooses layer 1 and 2 as additional skip connections (shown in dotted lines); (3) layer 4 chooses layer 3 as input and layer 2 as an additional skip connection.

We notice that different construction orders sometimes lead to the same network architecture, as illustrated in Figure 2. We put a constraint on the search space to mitigate this kind of duplication and accelerate the search procedure. Concretely, layer i must select its input from previous k layers, where k is set to be a small value. In this way, we favor the BFS-style construction manner in Figure 2a instead of Figure 2b. For example, if we set $k = 2$, the case in Figure 2b can be skipped because layer 4 cannot take layer 1 as input directly. In our experiments, we set $k = 5$ as a trade-off between expressiveness and search efficiency.

The tensor shape of the input word sequence is $<batch\_size, emb\_dim, max\_len>$, where $batch\_size$ is the pre-defined size of mini-batch; emb_dim is the embedding dimension of word vectors and max_len denotes the max length of the word sequence. In our implementation, we adopt a fixed-length representation, i.e., additional pad symbols are added to the tail if the input length is smaller than max_len; and the remaining text is discarded if the input length is larger than max_len. In all the layers, we keep the tensor shape as $<batch\_size, dim, max\_len>$, where dim is the dimension of hidden units. Note that dim may not equal to emb_dim, so an additional 1-D convolution layer is applied after the input layer.

After the network structure is built, the next step is to determine the options for each layer. In the search space, we incorporate four categories of candidate layers which are commonly used for text representation, namely *Convolutional Layers*, *Recurrent Layers*, *Pooling Layers*, and *Multi-Head Self-Attention Layers*. Each layer does not change the shape of input tensor, so one can freely stack more layers as long as the input shape is not modified.

Convolutional Layers. We define four kinds of 1-D convolution layers as candidate options with filter size 1, 3, 5, and 7 respectively. To keep the shape of output the same as input, we utilize the convolution of $stride = 1$ with SAME padding; and the number of output filters is equal to the input dimension. Note that the 1-D convolution with $filter\_size = 1$ and $stride = 1$ is analogue to a feed-forward layer. We apply Relu-Conv-BatchNorm once a convolutional layer is added.

Recurrent Layers. There are multiple kinds of recurrent layers, e.g., the vanilla RNN (Horne and Giles 1995), LSTM (Hochreiter and Schmidhuber 1997) and GRU (Bahdanau, Cho, and Bengio 2014). LSTM and GRU are known to be more advantageous than the vanilla RNN for capturing long-term dependencies in a text sequence; while GRU is usually several times faster than LSTM without loss of precision (Chung et al. 2014). Therefore, we leverage GRU layer as our RNN implementation. Specifically, we implement a bidirectional GRU that sums the output vectors of two opposite directions. One can also make LSTM and GRU as two candidate layers and let the search algorithm to make the decision.

Pooling Layers. The pooling layers calculate the maximum or average value within a filter window. We use pooling operations with SAME padding and $stride = 1$ so that the dimension of tensor does not change after pooling. For simplicity, we fix the filter size as 3 and only search between maximum or average pooling options. One can also enlarge the search space by allowing multiple choices of the filter size.

Multi-Head Self-Attention Layers. Multi-head self-attention layer is a major component in the neural network of Transformer (Vaswani et al. 2017). A Transformer block is constructed by one multi-head self-attention layer followed by one or more feed-forward layers. In our search space, we already have analogous to feed-forward layers, so we leverage the automatic search algorithm to decide how to combine them. The number of attention heads is set as 8 in all the experiments. We do not use positional embedding for the input of multi-head self-attention layers because it will destroy the translation invariance of succeeding pooling and CNN layers.

Search Algorithm

We leverage the ENAS (Efficient Neural Architecture Search) search algorithm (Pham et al. 2018) because it is one of most effective and efficient among all state-of-the-art search algorithms. ENAS searches for the best network architecture via reinforcement learning with weight sharing. In each step, the controller is responsible for sampling several child networks from the general search space. Then the child architectures are trained on the training set and evaluated on the validation set. The child networks share the same set of parameters with the global super-graph to accelerate the evaluation procedure. After the performance of each child network is evaluated, the accuracy is fed back to the controller and the parameters are updated through policy gradients based on REINFORCE (Williams 1992).

We reuse the open source code[2] of ENAS and implement the our novel search space accordingly. Concretely, the controller is implemented by a single LSTM layer, which generates the choice of each layer sequentially according to its topological order. For layer i, it first samples an input layer ID among $[max(0, i - k), i - 1]$ via softmax probabilities. Then it generates i binary outputs by sigmoid to identify if layer 0, 1, ..., $i - 1$ have skip connections with layer i. At last, an operator is selected for each layer. There are totally 8 options from 4 categories, i.e., 1-D convolution with filter
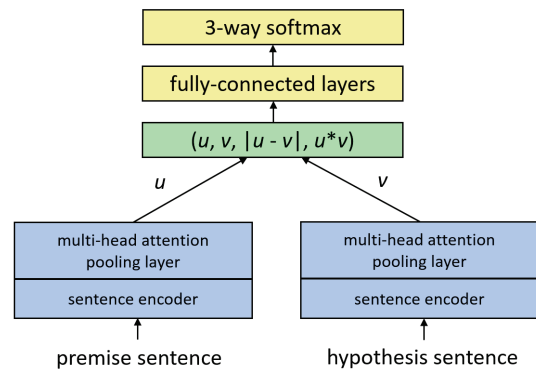
[2]https://github.com/melodyguan/enas

Figure 3: The sentence vector-based framework for natural language inference task

size 1, 3, 5, 7; max pooling; average pooling; Gated Recurrent Units (GRU) and multi-head self-attention. The selection probabilities of these options are calculated by softmax.

Tasks

We evaluate on two tasks to verify the feasibility and generality of our approach.

Text Classification is the task of assigning tags or categories to text according to its content. All layers in the text representation network are linearly combined (Peters et al. 2018) and followed by a max pooling layer and a fully connected layer with softmax activation to output the classification result.

Natural Language Inference is the task of determining whether a hypothesis sentence is entailment, contradiction or neutral given a premise sentence. We adopt the sentence vector-based framework (Bowman et al. 2015) for this task since our goal is to compare different text representation architectures. The framework is illustrated in Figure 3. The two sentences (i.e., hypothesis and premise) share the same text representation network, while the multi-head attention pooling layer (Chen, Ling, and Zhu 2018) is applied on top to generate the sentence embedding vector u and v. After that, we concatenate u, v, absolute element-wise distance $|u - v|$ and element-wise product $u \cdot v$ to construct the feature vector. We then feed the feature vector to three fully connected layers with ReLU activation before calculating 3-way softmax output.

Experiments

We first conduct neural architecture search and evaluate the performance on *SST*, a medium size dataset of text classification which has been extensively studied by human experts. Then we transfer the derived architectures to other text classification and natural language inference tasks.

Neural Architecture Search

SST is short for Stanford Sentiment Treebank (Socher et al. 2013) which is a commonly used dataset for sentiment classification. There are about 12 thousand reviews in *SST* and each review is labeled to one of the five sentiment classes.
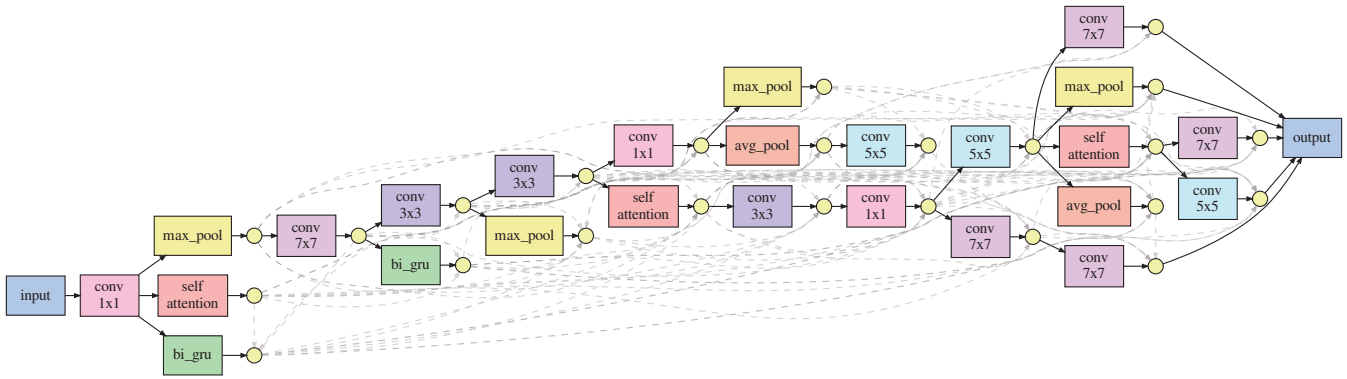
Figure 4: Visualization of TextNAS network: Rectangles represent layers, circles represent summations, one-way arrows represent inputs, and dotted one-way arrows represent skip connections.

Table 2: Statistics of text classification datasets

DATASET	#CLASS	#TRAIN	#VALID	#TEST
SST	5	8,544	1,101	2,210
SST-B	2	6,920	872	1,821
AG	4	120,000	-	7,600
SOGOU	5	450,000	-	60,000
DBP	14	560,000	-	70,000
YELP-B	2	560,000	-	38,000
YELP	5	650,000	-	50,000
YAHOO	10	1,400,000	-	60,000
AMZ	5	3,000,000	-	650,000
AMZ-B	2	3,600,000	-	400,000

There is another version of the dataset, *SST-Binary*, which has only two classes representing positive/negative while the neutral samples are discarded.

In our experiments, we perform 24-layers neural architecture search on *SST* dataset and evaluate the derived architectures on both *SST* and *SST-Binary* datasets. We follow the pre-defined train/validation/test split of the original datasets[3]. The word embedding vectors are initialized by pre-trained GloVe (glove.840B.300d[4]) (Pennington, Socher, and Manning 2014) and fine-tuned during training. We set the batch size as 128, max input length as 64, hidden unit dimension for each layer as 32, dropout ratio as 0.5 and L_2 regularization as 2×10^{-6}. We utilize Adam optimizer and learning rate decay with cosine annealing:

$$\lambda = \lambda_{min} + 0.5 \cdot (\lambda_{max} - \lambda_{min})(1 + cos(\pi T_{cur}/T)) \quad (1)$$

where λ_{max} and λ_{min} define the range of the learning rate, T_{cur} is the current epoch number and T is the cosine cycle. In our experiments, we set $\lambda_{max} = 0.005$, $\lambda_{min} = 0.0001$ and $T = 10$. After each epoch, ten candidate architectures are generated by the controller and evaluated on a batch of randomly selected validation samples. After training for 150 epochs, the architecture with the highest evaluation accuracy is chosen as the text representation network.

[3]https://nlp.stanford.edu/sentiment/code.html
[4]https://nlp.stanford.edu/projects/glove/

The whole process can be finished within 24 hours on a single Tesla P100 GPU. As visualized in Figure 4, the automatically discovered architecture is assembled by multiple paths and different categories of layers, including 13 convolution layers, 4 max-pooling layers, 2 average-pooling layers, 2 bi-directional GRU layers and 3 self-attention layers. Although it is much more complex than manual architectures, we still find that there are some design principles in line with human common-sense:

- The avg/max pooling layers and CNN/GRU/self-attention layers are alternatively stacked. The pooling layers help for extracting rotational/positional invariant features as inputs to other layers.

- There are convolution layers before and after each GRU and multi-head self-attention layers, which is similar to C-LSTM (Zhou et al. 2015) and Transformer (Vaswani et al. 2017). Intuitively, convolution operations generate local feature combinations (similar to n-grams) as complementary to GRU/self-attention layers which mainly capture long-term dependencies.

- The design principles look similar to InceptionV4 (Szegedy et al. 2017), which performs avg/max pooling and different convolution operations in parallel before aggregating them as final representation.

Result on SST

We evaluate the optimal result architecture by training it from scratch and searching for the best hyper-parameters. We set batch size as 128, max input length as 64, hidden unit dimension for each layer as 256. Other hyper-parameters are optimized by grid search on the validation data (showed in the appendix). We compare our architecture with state-of-the-art networks designed by human experts, including 24-layers Transformer which is the text representation architecture leveraged in BERT (Devlin et al. 2018). We also compare to the original search spaces defined in ENAS (Pham et al. 2018):

- **ENAS-MACRO** is a macro search space over the convolutional and pooling layers, which is originally designed for image classification tasks. There are 6 operations in

Table 3: Results on SST dataset. For each dataset, we conduct significance test against the best reproducible model, and * means that the improvement is significant at 0.05 significance level.

MODEL	SST	SST-B
LAI ET AL., 2015	47.21	-
ZHOU ET AL., 2015	49.20	87.80
LIU ET AL., 2016	49.60	87.90
TAI ET AL., 2016	51.00	88.00
KUMAR ET AL., 2016	**52.10**	**88.60**
24-LAYERS TRANSFORMER	49.37	86.66
ENAS-MACRO	51.55	**88.90**
ENAS-MICRO	47.00	87.52
DARTS	**51.65**	87.12
SMASH	46.65	85.94
ONE-SHOT	50.37	87.08
RANDOM SEARCH	49.20	87.15
TEXTNAS	**52.51**	**90.33***

the search space: convolutions with filter sizes 3×3 and 5×5, depthwise-separable convolutions with filter sizes 3×3 and 5×5 (Chollet 2017), max pooling and average pooling of kernel size 3×3. In our experiments, we search for a macro neural network consisting of 24 layers.

- **ENAS-MICRO** is a micro search space over normal and reduction cells. There are two kinds of cells, i.e., normal cells and reduction cells. In each cell, there are $B = 10$ nodes, where node 1 and node 2 are treated as the inputs of current cell. For each of the remaining $B - 2$ nodes, the RNN controller makes two decisions: 1) selecting two previous nodes as inputs to the current node and 2) selecting two operations to apply on the input nodes. There are 5 available operations: identity, separable convolution with kernel size 3×3 and 5×5, average pooling and max pooling with kernel size 3×3. In our experiments, we stack the cells for 6 times. The normal cells and reduction cells are stacked alternatively.

We also compare to other search algorithms which have similar time complexities as ENAS, including DARTS (Liu, Simonyan, and Yang 2018), SMASH (Brock et al. 2017), One-Shot (Bender et al. 2018) and Random Search with Weight Sharing (Li and Talwalkar 2019). Unless specified, we utilize the default settings of their open-source codes without tuning the hyper-parameters or modifying the proposed search spaces except for replacing all 2-D convolutions with 1-D (detailed settings can be found in the appendix).

The evaluation results are shown in Table 3. We can see that the neural architecture discovered by TextNAS achieves competitive performances compared with state-of-the-art manual architectures, including the 24-layers Transformer adopted by BERT. At the same time, it outperforms other network architectures discovered automatically by other search spaces and algorithms. Specifically, the accuracy is improved by 11.7% from ENAS-MICRO and 1.9% from ENAS-MACRO on the SST dataset respectively, which shows the

superiority of our novel search space for text representation. It should be noticed that there are other publications that have reported higher accuracies. However, they are not directly comparable to our scenario since they incorporate various kinds of external knowledge, e.g., BERT (Devlin et al. 2018) pre-trains on a large external corpus and (Yu et al. 2017) exploits syntax information in the Tree-LSTM model.

Result on Architecture Transfer

Text Classification We transfer the derived architecture as text representation networks to other eight text classification datasets[5] (Zhang, Zhao, and LeCun 2015). These datasets are from various domains including sentiment analysis, Wikipedia article categorization, news categorization and topic classification. The counts of samples are widely spread from hundreds of thousands to several millions as summarized in Table 2.

We follow the train/test split of the original datasets in all our experiments. For those datasets without validation set, we randomly select 5% samples from the training set as validation data. For all datasets, we use pre-trained GloVe embedding to initialize word vectors and fine-tune them during training. To simplify the learning rate fine-tuning procedure for different datasets, we adopt an auto-decay strategy instead of cosine annealing. Given an initial learning rate, we use a small learning rate ($0.1 \times init\_rate$) to warm up the training procedure for 5 epochs; then we start from $init\_rate$ and decay it with a factor of 0.2 when the average validation accuracy of 7 recent epochs on the validation data drops. Finally, after 4 times of decay, we update the model for another 6 epochs on the full training set (training + validation). As a result, only one hyper-parameter, i.e., $init\_rate$, is required for each dataset. For critical hyper-parameters, we employ grid search on the validation data. Specifically, we search in $\{0.08, 0.05, 0.02\}$ for learning rate, $\{64, 128\}$ for batch size, $\{64, 256, 512\}$ for max input length, $\{2 \times 10^{-9}, 2 \times 10^{-7}, 1 \times 10^{-6}, 2 \times 10^{-6}\}$ for L_2 regularization, $\{0.0, 0.2, 0.5\}$ for drop-out ratio, and $\{32, 64, 128, 256\}$ for hidden units dimension respectively. We observe that the Adam optimizer is not stable in several settings, so we adopt stochastic gradient descent with momentum 0.9 for training on all the datasets. More detailed settings are described in the appendix.

The test accuracies on all datasets are shown in Table 4. The results demonstrate that the TextNAS model outperforms state-of-the-art methods on all text classification datasets except Sogou. One potential reason is that Sogou is a dataset in Chinese language, while the Glove embedding vectors are trained by English corpus. One can improve the performance by adding Chinese-language embeddings or char-embeddings, but we do not add them to keep the solution neat. In addition, we can pay a specific attention to the comparison of TextNAS with 29-layers CNN (Conneau ET AL., 2016) and 24-layers Transformer (VASWANI ET AL., 2017). As shown in the table, the TextNAS network improves two baselines by a large margin, indicating the advantage for mixture of different layers.

---

[5]The datasets are available at http://xzh.me/

Table 4: Test accuracy on the text classification datasets. For each dataset, we conduct significance test against the best reproducible model, and * means that the improvement is significant at 0.05 significance level.

MODEL	AG	SOGOU	DBP	YELP-B	YELP	YAHOO	AMZ	AMZ-B
ZHANG ET AL., 2015	92.36	**97.19**	98.69	95.64	62.05	71.20	59.57	95.07
JOULIN ET AL., 2016	**92.50**	96.80	98.60	95.70	63.90	72.30	60.20	94.60
CONNEAU ET AL., 2016	91.33	96.82	98.71	**95.72**	**64.72**	**73.43**	**63.00**	**95.72**
24-LAYERS TRANSFORMER	92.17	94.65	**98.77**	94.07	61.22	72.67	62.65	95.59
ENAS-MACRO	92.39	96.79	**99.01**	**96.07**	64.60	73.16	62.64	**95.80**
ENAS-MICRO	92.27	**97.24**	99.00	96.01	64.72	70.63	58.27	94.89
DARTS	92.24	97.18	98.90	95.84	65.12	73.12	62.06	95.48
SMASH	90.88	96.72	98.86	95.62	**65.26**	**73.63**	**62.72**	95.58
ONE-SHOT	92.06	96.92	98.89	95.78	64.78	73.20	61.30	95.20
RANDOM SEARCH	**92.54**	97.13	98.98	96.00	65.23	72.47	60.91	94.87
TEXTNAS	**93.14**	96.76	99.01	96.41*	66.56*	73.97*	63.14*	95.94*

Natural Language Inference We carry out experiments on two Natural Language Inference (NLI) datasets by leveraging the network architecture of TextNAS as sentence encoder. The *SNLI* dataset[6] (Bowman et al. 2015) consists of 549,367 samples for training, 9,842 samples for validation and 9,824 samples for testing. The *MultiNLI* dataset[7] (Williams, Nangia, and Bowman 2018) contains 392,702 pairs for training. It has two separate sets for evaluation: *MNLI-M* (matched set) has 9,815 pairs for validation and 9,796 pairs for testing; *MNLI-MM* (mismatched set) contains 9,832 pairs for validation and 9,847 pairs for testing. Each sample is labeled with one of three labels: entailment, contradiction and neutral.

We initialize the word embedding layer by the concatenation of pre-trained GloVe embeddings and charNgram embeddings (Hashimoto et al. 2016). The word embedding vectors are fine-tuned during training. The outputs of all layers in the sentence encoder are linearly combined to produce the vector-based representation. We set the dimension of hidden units as 512 for all layers in the sentence encoder and 2400 for the fully connected layers before softmax output. Dropout is adopted on the output of each word-embedding, GRU and fully connected layer. Adam optimizer with learning rate decay strategy of cosine annealing is utilized to train the model. Detailed settings are optimized by grid search and presented in the appendix.

The evaluation results are illustrated in Table 5. To get a fair comparison, we only compare with state-of-the-art sentence vector-based models that perform classification on the sole basis of a pair of fixed-size sentence representations. As shown in the table, TextNAS achieves competitive test accuracy on both *SNLI* and *MNLI* datasets consistently. In addition, it performs much better than the 24-layer Transformer, which verifies the effectiveness of our search space and methodology.

To conclude, TextNAS generates novel and transferable network architecture for text classification and natural language inference tasks. By searching neural architectures on a

Table 5: Results on NLI datasets. For each dataset, we conduct significance test against the best reproducible model, and * means that the improvement is significant at 0.05 significance level.

MODEL	SNLI	MNLI-M/MM
NIE AND BANSAL, 2017	86.0	**74.6** / 73.6
IM AND CHO, 2017	86.3	74.1 / 72.9
TALMAN ET AL., 2018	86.6	73.7 / 73.0
CHEN ET AL., 2018	86.6	73.8 / **74.0**
KIELA ET AL., 2018	**86.7**	-
24-LAYERS TRANSFORMER	85.2	70.4 / 70.2
TEXTNAS	87.4*	**74.9** / 74.2

relatively small dataset and then transferring it to larger ones, the network design procedure can be performed efficiently and effectively.

Conclusion & Future Work

In this paper, we propose a novel architecture search space specialized for text representation by leveraging multi-path ensemble and a mixture of convolutional, recurrent, pooling, and self-attention layers. We demonstrate that by applying an efficient search algorithm, the TextNAS neural network architecture achieves state-of-the-art performance in various text-related applications. In addition, the architecture is explainable and transferable to other tasks. Future work mainly falls into three aspects: (1) uniting neural architecture search with state-of-the-art transfer learning frameworks, e.g., BERT; (2) exploring search acceleration techniques and conduct neural architecture search on larger datasets; (3) applying the TextNAS framework to other text-related tasks, such as Q&A, machine translation and search relevance.

References

Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

---

[6]https://nlp.stanford.edu/projects/snli/

[7]https://www.nyu.edu/projects/bowman/multinli/

Baker, B.; Gupta, O.; Raskar, R.; and Naik, N. 2017. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*.

Bender, G.; Kindermans, P.-J.; Zoph, B.; Vasudevan, V.; and Le, Q. 2018. Understanding and simplifying one-shot architecture search. In *ICML*, 549–558.

Bowman, S. R.; Angeli, G.; Potts, C.; and Manning, C. D. 2015. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*.

Brock, A.; Lim, T.; Ritchie, J. M.; and Weston, N. 2017. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*.

Chen, Q.; Ling, Z.-H.; and Zhu, X. 2018. Enhancing sentence embedding with generalized pooling. In *COLING*. Santa Fe, USA: ACL.

Chollet, F. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1251–1258.

Chung, J.; Gulcehre, C.; Cho, K.; and Bengio, Y. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Conneau, A.; Schwenk, H.; Barrault, L.; and Lecun, Y. 2016. Very deep convolutional networks for text classification. *arXiv preprint arXiv:1606.01781*.

Conneau, A.; Kiela, D.; Schwenk, H.; Barrault, L.; and Bordes, A. 2017. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

dos Santos, C., and Gatti, M. 2014. Deep convolutional neural networks for sentiment analysis of short texts. In *COLING*, 69–78.

Hashimoto, K.; Xiong, C.; Tsuruoka, Y.; and Socher, R. 2016. A joint many-task model: Growing a neural network for multiple nlp tasks. *arXiv preprint arXiv:1611.01587*.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

Horne, B. G., and Giles, C. L. 1995. An experimental comparison of recurrent neural networks. In *NIPS*, 697–704.

Im, J., and Cho, S. 2017. Distance-based self-attention network for natural language inference. *arXiv preprint arXiv:1712.02047*.

Kalchbrenner, N., and Blunsom, P. 2013. Recurrent convolutional neural networks for discourse compositionality. *arXiv preprint arXiv:1306.3584*.

Li, L., and Talwalkar, A. 2019. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*.

Liu, X.; Gao, J.; He, X.; Deng, L.; Duh, K.; and Wang, Y.-Y. 2015. Representation learning using multi-task deep neural networks for semantic classification and information retrieval.

Liu, C.; Zoph, B.; Shlens, J.; Hua, W.; Li, L.-J.; Fei-Fei, L.; Yuille, A.; Huang, J.; and Murphy, K. 2017. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559*.

Liu, H.; Simonyan, K.; and Yang, Y. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Luong, M.-T.; Pham, H.; and Manning, C. D. 2015. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

Nie, Y., and Bansal, M. 2017. Shortcut-stacked sentence encoders for multi-domain inference. *arXiv preprint arXiv:1708.02312*.

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

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2018. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A.; and Potts, C. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, 1631–1642.

Szegedy, C.; Ioffe, S.; Vanhoucke, V.; and Alemi, A. A. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, 12.

Tai, K. S.; Socher, R.; and Manning, C. D. 2015. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

Talman, A.; Yli-Jyrä, A.; and Tiedemann, J. 2018. Natural language inference with hierarchical bilstm max pooling architecture. *arXiv preprint arXiv:1808.08762*.

Tang, D.; Qin, B.; and Liu, T. 2015. Document modeling with gated recurrent neural network for sentiment classification. In *EMNLP*, 1422–1432.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *NIPS*, 5998–6008.

Williams, A.; Nangia, N.; and Bowman, S. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 1112–1122. Association for Computational Linguistics.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8(3-4):229–256.

Yang, Z.; Yang, D.; Dyer, C.; He, X.; Smola, A.; and Hovy, E. 2016. Hierarchical attention networks for document classification. In *NAACL*, 1480–1489.

Yoon, D.; Lee, D.; and Lee, S. 2018. Dynamic self-attention: Computing attention over words dynamically for sentence embedding. *arXiv preprint arXiv:1808.07383*.

Yu, L.-C.; Wang, J.; Lai, K. R.; and Zhang, X. 2017. Refining word embeddings for sentiment analysis. In *EMNLP*, 534–539.

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

Zhou, C.; Sun, C.; Liu, Z.; and Lau, F. 2015. A c-lstm neural network for text classification. *arXiv preprint arXiv:1511.08630*.

Zoph, B., and Le, Q. V. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2017. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012* 2(6).