NASABN: A Neural Architecture Search Framework for Attention-Based Networks

1st Kun Jing
University of Chinese Academy of Sciences
Beijing, China
jingkun18@mails.ucas.ac.cn

2nd Jungang Xu*
University of Chinese Academy of Sciences
Beijing, China
xujg@ucas.ac.cn

3rd Hui Xu
Zugeng Technology
Beijing, China
huixu1024@hotmail.com

***Abstract*—Recently, neural architecture search (NAS) has emerged as a technique of growing concern in automatic machine learning (AutoML). Meanwhile, attention-based models, such as attention-based recurrent neural network, transformer-based model, etc., have been widely used in deep learning applications. However, there is no efficient NAS method that can search the architecture of attention-based model so far. To solve this problem, we propose a framework named neural architecture search for attention-based networks (NASABN) by abstracting attention-based models and extracting undefined parts of the model, including the attention layers and cells. NASABN is flexible and general enough to fit different NAS methods, which can also be transferred across different datasets. We conduct extensive experiments with NASABN using gradient descent-based methods like DARTS on Penn Treebank (PTB) and WikiText-2 (WT2) datasets respectively, and achieve competitive performance compared with the state-of-the-art methods.**

***Index Terms*—neural architecture search, recurrent neural network, language modeling, attention-based model**

I. Introduction

In recent years, the application of deep learning in various fields has made significant breakthroughs, such as natural language processing (NLP) and computer vision (CV). Different architectures [1]–[4] are proposed for improving performance. Novel neural architecture plays an important role in this development [5], [6]. However, neural architectures are currently designed primarily by human experts, just like features designed by human experts before deep learning, which is an extremely time-consuming task that requires much expert knowledge. Therefore, NAS has attracted great attention.

Many NAS methods have been proposed. Many works [7]–[13] have tried different NAS methods, including random search, Bayesian optimization, evolutionary algorithm, reinforcement learning, and gradient-based methods. In recent years, attention-based models [4], [14], [15] are on the rise, which have been proved to achieve better performance. However, these above NAS methods only conduct architecture search for classic recurrent neural networks (RNNs) or convolutional neural networks (CNNs). So far, architecture search for attention-based neural networks has been rarely studied.

In our work, a neural architecture search framework for attention-based networks is proposed, which applies the existing NAS methods like ENAS, DARTS into our proposed framework to search the attention-based neural architecture. We mainly introduce how to extend DARTS [13] to conduct architecture search for attention-based RNN. This framework can be applied for different attention-based networks using different search methods, which is thereby flexible and general.

In our experiments, as previous works [7], [8], [13], we use language modelling task to evaluate the discovered architecture of attention-based RNN. Our work discovers a recurrent cell with 9 nodes, which achieves a competitive result (test perplexity of 56.5 on PTB and test perplexity of 71.8 on WT2), which outperforms some models and reduces computational complexity.

In summary, our contributions are as follows:

- We propose a general neural architecture search framework for attention-based networks. Specifically, we introduce DARTS method into the framework to search attention-based RNN architecture.
- We achieve competitive results (test perplexity of 56.5) on PTB compared to existing methods. Meanwhile, we also achieved test perplexity of 71.8 on WT2.
- We find a problem of DARTS that it tends to select the operations that need not learn, and we suggest to use phased training to solve the problems.

II. Related Work

A. Grid Search

Grid search is the most traditional method for hyper-parameter optimization [1]. Grid search [16] is the method that divides the whole space of hyper-parameters into grids, then trains the model for all values on the grid for the best hyper-parameters. To avoid bad points of the space of hyper-parameters, Reference [17] proposed coarse grid search first, and then use fine grid search in a better region. Besides, a contracting-grid search [18] is proposed, i.e., on the better point of the old grid, creating a new grid that reduces to half of the size of the old one.

B. Random Search

Random Search is the method that selects a set of hyper-parameters at random to verify whether it is a better one.

* Corresponding Author

[1]Neural architecture is regarded as a special hyper-parameter. Hyper-parameter optimization can be used into NAS.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

Random search [8], [13], [19] has been proved to be a common and efficient method.

C. Reinforcement Learning Based Methods

The first NAS method [7] is based on reinforcement learning, which uses an RNN trained by reinforcement learning to generate neural architecture. Almost at the same time, MetaQNN [20] was introduced, which trains the learning agent to sequentially choose CNN layers by Q-learning with an ϵ-greedy exploration strategy and experience replay. Searching for the whole architecture is difficult and time-consuming. Some cell-based methods were proposed, such as NASNet [9], ENAS [8], and BlockQNN [21]. The difference is that the cell architecture is searched firstly, and then they stack multiple cells to form the whole network.

D. Evolutionary Algorithm Based Methods

Evolutionary algorithm takes inspiration from biological evolution. Genetic CNN [22] uses binary encoding scheme to represent the neural architecture and Russian roulette process for selection. To represent the variable-length architecture, a Cartesian genetic programming encoding scheme [23] is proposed for representation, which uses the modified $(1+\lambda)$ evolutionary strategy; others are the same as standard Cartesian genetic programming. [24] encodes the neural architecture as a graph, in which vertices represent tensors or activations and edges represent neural connections. Its DNA encoding mutates on a predefined set that changes the graph, and then they use tournament selection. A novel hierarchical encoding scheme [25] was introduced, which imitates the modularized design pattern commonly adopted by human experts. Tournament selection has been widely applied for selection, such as [11], [25], [26]. Furthermore, tournament selection was improved by introducing an age property to favour the younger genotypes [11].

E. Bayesian Optimization

Different from grid search, random search, and evolutionary algorithom based search, Bayesian optimization takes advantage of the historical information of the settings of hyper-parameters by a probability model of the objective function. Bayesian optimization is divided into three categories according to the probability model, including Random Forests [27], Gaussian Processes [28], and Tree Parzen Estimators [29]. Gaussian processes-based model is the most commonly used. To accelerate hyper-parameter optimization, a Bayesian optimization procedure [30] called FABOLAS was constructed, in which loss and training time are defined as the functions of dataset size, and it automatically trades off high information gain about the global optimum against computational cost. A practical hyper-parameter optimization method [31] was proposed for combining the benefits of both Bayesian optimization and bandit-based methods to balance two aspects, i.e., strong performance and fast convergence.

F. Gradient Descent Based Methods

Reference [32] suggested to directly compute exact gradients of cross-validation performance concerning all hyper-parameters by chaining derivatives backward through the entire training procedure. A fabric [33] was proposed, which embeds an exponentially large number of architectures. The fabric consists of a 3D trellis that connects response maps at different layers, scales, and channels with a sparse homogeneous local connectivity pattern. A continuous hyper-parameter optimization algorithm using inexact gradient information was proposed in [34]. An approach in [35] uses residual blocks as the modular components, predefines the input connections and aggregation pathways of each branch, and uses gates to determine the final connectivity. A method in [36] transforms a discrete neural network architecture space into a continuous and differentiable one, which enables the use of standard gradient-based optimization techniques. Inspired by the above works, DARTS [13] uses the continuous relaxation of the architecture representation by placing a mixture of candidate operations on each edge, which allows to efficiently search architectures using gradient descent for RNN and CNN. Differentiable Hyper-Parameter Grid Search and Hyper-Cuboid search space [37] were proposed for more general parameter optimization.

Our work is significantly different from the above works. We proposed a general framework for searching architectures of attention-based networks rather than a specific search method, while the above works just propose NAS methods for classic RNN and CNN. The NAS search methods can be used in our framework to achieve the architecture search of attention-based models. Since DARTS is a widely used baseline, we only use it as a NAS method in our framework for verification.

III. Methodology

A. Attention Based Network Search

The related works [7], [8], [13] regard the NAS task as searching architecture parameter α for network connectivity and types of operations on connections. The goal of the NAS task is to find the best architecture parameter α^*, which can be formulated as an optimization problem:

$$\alpha^* = argmin_\alpha \; L_{val}(\alpha, w^*(\alpha)). \tag{1}$$

Considering that attention-based networks are the classic RNNs or CNNs combined with the attention layer, we propose a general framework for all attention-based networks as illustrated in Fig. 1. The architecture $\alpha' = \{\alpha, \alpha_{att}\}$ of attention-based networks depend on two aspects: the neural architecture parameter α and the attention type parameter α_{att}. The goal of NAS task changes to find the best architecture parameter α'^*. The optimization problem (as shown in Formula (1)) can be rewritten as

$$\alpha'^* = argmin_{\alpha'} \; L_{val}(\alpha, \alpha_{att}, w^*(\alpha, \alpha_{att})). \tag{2}$$

There are two options of the framework:

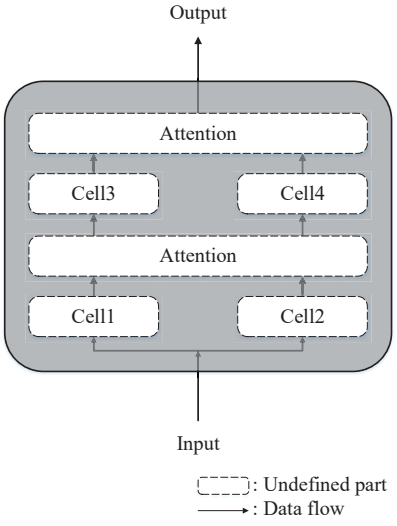

Fig. 1. The architecture of the attention-based model. This also is the graphical description of the neural architecture search framework for attention-based networks. All non-solid lines are undefined, including the cells and the attention layers. During architecture search, we only concern about the undefined parts exposed from the black box.

- If the attention layer is defined, the whole architecture is definite;
- If the attention layer is not defined, the type of attention layer can be learned together with cell architectures.

For simplification, we take attention-based RNN as an example of our framework in detail.

Following the attention-based model [14], we represent an RNN with attention layers as shown in Fig. 2. Furthermore, the existing NAS methods, including the methods mentioned in the related works, can be easily applied to our framework. When the attention layer is not defined, the choice of attention should be converted to the corresponding search space and learned together with the cell architecture in the same way.

In this work, we apply DARTS into our framework as follows. To search attention based RNN, we convert the choices of attention to continuous space using the parameter $\alpha_{att} \in R^{n_{att}}$ that implies the type of attention, where the candidate attention layer can include simple global attention without learnable parameters, masked multi-head self-attention layer, and others. n_{att} is the number of types of attention. It is noted that α_{att} is learned in the same way as other architecture parameters α. This can be formulated as

$$o_{att} = \sum_{i}^{n_{att}} \frac{exp(\alpha_{att}^{i})}{\sum_j exp(\alpha_{att}^{j})} Att_i(RNN(x_1^n)). \tag{3}$$

At the end of the training, we get the best attention $Att^* = Att_i$ where i is the position of maximum in α_{att}, i.e.,

$$i = argmax_i \ \alpha_{att}^{i}. \tag{4}$$

B. Cell Architecture Search

Following the previous NAS methods [9], [11], [13], we search the neural cell as the block of the whole architecture.

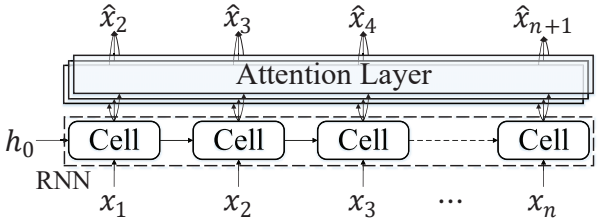

Fig. 2. The architecture of attention-based RNN. The current cell receives the output of the previous cell and feature representation at the current time step as inputs. The outputs of all cells are sent to the attention layer to get the prediction at each time step.

The whole architecture is introduced in the previous subsection. The architecture of attention-based RNN is shown in Fig. 2.

RNN cell regards the hidden state of the previous time step and the input feature vector of the current time step as the inputs, and then generates the hidden state of the current time step as the output. As shown in Fig. 2, the searched cells are connected over time steps to form the RNN; and then an attention layer is stacked on the last layer of RNN.

As shown in Fig. 3, we represent a cell as a directed acyclic graph (DAG) consisting of an ordered sequence of N nodes. In the DAG, each node i represents a feature representation r_i, such as a feature vector. Each directed edge (i, j) represents a mixed function $f_{(i,j)}$ that transforms r_i. Each intermediate node can be computed using all of its predecessors, i.e.,

$$r_j = \sum_{i<j} f_{(i,j)}(r_i), \tag{5}$$

where $f_{(i,j)}$ is the weighted sum of all possible functions

$$f_{(i,j)}(r_i) = \sum_{f \in F} \frac{exp(\alpha_{(i,j)}^{f})}{\sum_{f' \in F} exp(\alpha_{(i,j)}^{f'})} f(r_i). \tag{6}$$

$\alpha_{(i,j)}^{f}$ is the architecture parameter for the function f on the edge (i, j). F is the candidate function set for RNN, which contains 5 functions: (1) zero operation; (2) identity mapping; (3) linear transformation followed by tanh activation; (4) linear transformation followed by sigmoid activation; (5) linear transformation followed by relu activation.

C. Differentiable Method

We search architectures by learning a set of continuous variables α' that converts discrete search space to continuous search space. To jointly learning the architecture α' and the weights w using gradient descent, this is regarded as a bi-level optimization problem,

$$min_{\alpha'} \quad L_{val}(w^*(\alpha'), \alpha') \tag{7}$$

$$s.t. \quad w^*(\alpha') = argmin_w \ L_{train}(w, \alpha'), \tag{8}$$

where L_{train} and L_{val} are the training and validation loss respectively, which are hard to be optimized.

We use the approximation scheme and the one-step unrolled learning objective [13] to accelerate the optimization, as shown

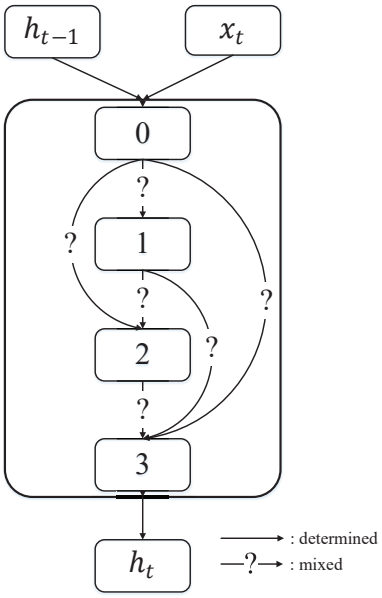

Fig. 3. Search space of the RNN cell. The function on each edge is unknown initially. Each edge is a function that is the weighted sum of each one in the candidate functions F. After completing architecture search, the function on each edge is determined using the most likely functions, i.e., $f_{(i,j)} = argmax_{f \in F} \alpha^f_{(i,j)}$.

in Algorithm 1. At the end of the training, we greedily choose the most possible set of operations and get the final architecture.

Algorithm 1 Gradient-Based Architecture Search

Create the whole architecture using the cell with mixed functions on edges.
while not converged **do**
  Update architecture α' by gradient descent:
  $$\nabla_{\alpha'} L_{val}(w - \xi \nabla_w(w, \alpha'), \alpha')$$
  Update weights w by gradient descent:
  $$\nabla_w L_{train}(w, \alpha')$$
end while
Derive the learned architecture based on the α' greedily.

IV. Experiments

To explore our architecture search framework, we use language modelling task to evaluate searched attention-based RNN. We search architectures on PTB corpus, and evaluate the architecture on PTB and WT2. All our models are trained on a server with Intel(R) Xeon(R) CPU E5-2640 v4 @ 2.40GHz and NVIDIA TESLA P100 16GB GPUs.

A. Datasets

a) Penn Treebank (PTB): PTB [38] is the portion of the Wall Street Journal corpus. We preprocess PTB following [39]. PTB corpus consists of over 1M words (including 929,589 words in the training set, 73,760 words in the validation set, and 82,430 words in the test set), and the vocabulary size is 10,000. PTB is commonly used in language modelling task.

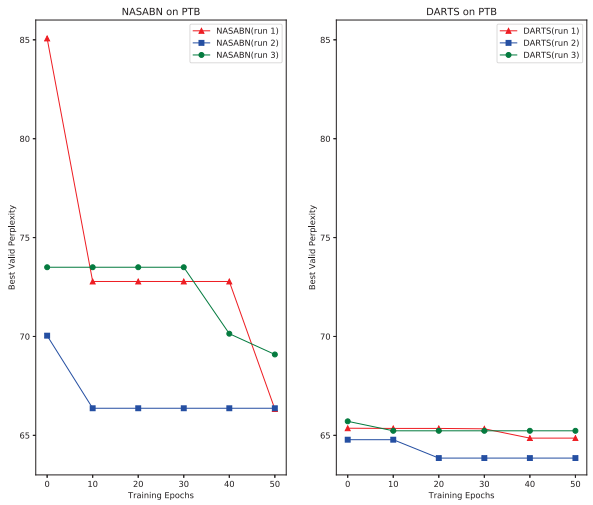

Fig. 4. Search process of NASABN and DARTS on PTB. We train NASABN and DARTS with 50 epochs on PTB respectively and evaluate the discovered architectures per 10 epochs. The evaluation method is to retrain the discovered architectures with 300 epochs on the training set. Architecture search run three times, i.e., 2 $models \times 3$ $runs \times 6$ $evaluations/(model \cdot run)$ architectures are evaluated. For each run, we report the best valid perplexity over epoch.

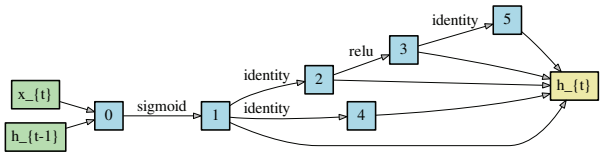

Fig. 5. The RNN cell learned on PTB.

b) WikiText-2 (WT2): WT2 [40] was constructed using text extracted from Wikipedia. The vocabulary size is 33,278. The training set contains 600 articles with 2,088,628 tokens. The validation set contains 60 articles with 217,646 tokens. The test set contains 60 articles with 245,569 tokens.

B. Architecture Search

The cell of RNN consists of $N = 9$ nodes (i.e., 2 input nodes, 6 intermediate nodes, and 1 output node). Following ENAS [8] and DARTS [13], the first intermediate node is obtained by summing over the linear transformations of the two input nodes and then using a $tanh$ activation function on it. Other intermediate nodes are calculated by learned edges or functions. All the intermediate nodes, except for the first one, are averaged and sent to the output node. As in ENAS and DARTS, we use the highway technology for each function, and batch normalization for each node to prevent gradient explosion. The cell is applied in the whole architecture as Fig. 2. For simplification, we only use the single layer with the cell and the definite attention layer that use global dot attention [48].

In our experiments, the hyper-parameters setting on PTB is the same as DARTS. Both embedding and hidden sizes are set to 300 for training in a single GPU. The model is trained for 50 epochs with batch size of 256, BPTT length

TABLE I
COMPARISON WITH DIFFERENT LANGUAGE MODELS ON PTB.

Architecture	Perplexity		Params	Search Cost
	valid	test	(M)	(GPU days)[a]
LSTM + *att* + *select* + *entropy* [41][b]	-	131.4	-	-
RMN [42]	-	123.3	-	-
RMR [42]	-	134.3	-	-
Variational RHN [43]	67.9	65.4	23	-
LSTM [39]	60.7	58.8	24	-
LSTM + skip connections [44]	60.9	58.3	24	-
LSTM + 15 softmax experts [45]	58.1	56.0	22	-
NAS [7]	-	64.0	25	10^4 CPU days
ENAS [8]	60.8	58.6	24	0.5
DARTS (first order) [13]	60.2	57.6	23	**0.13**
DARTS (second order) [13]	**58.1**	**55.7**	23	0.25
GDAS [46]	59.8	57.5	23	0.4
Random search baseline[c]	60.3	57.8	**18**	2
NASABN (first order)	58.9	56.5	**18**	0.15

[a]Different from DARTS [13], search cost here is the time it takes to run a single search, which does not include the cost of architecture evaluation.
[b]In this model, *att* denotes the attention; *select* denotes the memory selection mechanism; *entropy* denotes the entropy regularization.
[c]The best architecture among 8 samples according to the validation perplexity after 300 training epochs.

TABLE II
COMPARISON WITH DIFFERENT LANGUAGE MODELS ON WT2.

Architecture	Perplexity		Params	Search Cost
	valid	test	(M)	(GPU days)[a]
LSTM + augmented loss [47]	91.5	87.0	28	-
LSTM [39]	69.1	66.0	33	-
LSTM + skip connections [44]	69.1	65.9	**24**	-
LSTM + 15 softmax experts [45]	66.0	63.3	33	-
ENAS (searched on PTB) [8]	72.4	70.4	33	0.5
DARTS (searched on PTB) [13]	71.2	69.6	33	0.25
GDAS (searched on PTB) [46]	**71.0**	**69.4**	33	0.4
NASABN (searched on PTB)	75.0	71.8	30	**0.15**

[a]Search cost is consistent with Table I.

of 35, gradient clipping of 0.25. Two optimizers are used to optimize architecture parameter α' and model parameter w respectively. The optimizer for architecture parameter optimization is Adam with initial learning rate of 3×10^{-3}, weight decay of 10^{-3}, momentum $\beta = (0.9, 0.999)$. The optimizer for model parameter optimization is SGD without momentum, with learning rate of 20, weight decay of 5×10^{-7}. Besides, for regularization, we apply variational dropouts of 0.2 to word embeddings, 0.75 to the cell input, 0.25 to all the hidden nodes, a dropout of 0.75 to the output layer, and slowness regularization of 0.001 on RNN activation. The training takes less than 4 hours on a single P100 GPU.

C. Architecture Evaluation

To evaluate the final architectures, we retrained the attention-based RNN with the discovered cell as shown in Fig. 5 on train sets of PTB and WT2 respectively. It is noted that batch normalization is disabled during architecture evaluation. Because of initialization sensitivity according to DARTS [13], we search architecture three times with different random seeds as shown in Fig. 4. Then we report the architecture of which final valuation performance is the best (Fig. 5) for PTB and list test performance for PTB and WT2.

a) Evaluation on PTB: The attention-based RNN is trained with batch size of 64 using averaged SGD (ASGD), with learning rate of 20 and weight decay of 8×10^{-7}. Following DARTS, we start with SGD and trigger ASGD using the same protocol. Both embedding and hidden sizes are set to 850; the token-wise dropout on the embedding layer is set to 0.1. Other hyper-parameters are the same as those for architecture search. The training takes 9 days on a single P100 GPU.

b) Evaluation on WT2: Except that embedding and hidden sizes are set to 700, weight decay is set to 5×10^{-7}, and variable dropout is set to 0.15 for hidden nodes, other hyper-parameters are the same as those for architecture evaluation on PTB. The training takes 9 days on a single P100 GPU.

D. Results Analysis

The search process of NASABN is shown in Fig. 4. To compare with DARTS, we also show the search process of DARTS using the code provided by [13]. Although the initial points of NASABN are worse than those of DARTS, its per-

plexity can effectively decrease later. NASABN is sensitive to initialization. The discovered cell architecture for the attention-based network is shown in Fig. 5.

The experimental results on PTB are reported in Table I. NASABN achieves test perplexity of 56.5 on PTB, which is better than most manually designed and automatically searched architectures (including the searched architecture randomly in our search space). Our random search baseline model has almost the same performance as DARTS using first order optimization method, which shows that our search space is competitive to the search space of DARTS. Noted that our model has fewer parameters and less cost due to fewer intermediate nodes of which the size remains unchanged instead of the framework itself. This is because NASABN introduces attention mechanism, which reduces the complexity of the model. The experimental results prove the effectiveness of NASABN.

Table II shows the result of the transferred model from PTB to WT2. The model achieves test perplexity of 71.8 on WT2, which is not a good result. The main reason is that we utilize the first order optimization method, which has no advantage over second order optimization. In addition, PTB and WT2 have different distributions of linguistic phenomena and different numbers of words. Directly searching the architecture on the task of interest is an appropriate solution.

During architecture search, we obtain the similar finding in the previous works [6], [49] that DARTS tends to learn the operations that need not learn, such as zero operation for RNN, zero operation and pooling operation for CNN. In the initial stage when the weights are randomly initialized, these operations have greater advantages than those with learnable weights, which is because those operations with learnable weights have not learned useful feature representations yet. We recommend a phased training to solve this problem, i.e. in the first stage, the mixed network is trained for a suboptimal network, and in the second stage, the architecture is searched in the suboptimal network.

V. Conclusion and Future Work

We proposed a general neural architecture search framework called NASABN for attention-based networks, which can search the attention layers of the model in the same way as cell architecture search. We can apply different NAS methods in the framework to search the optimal architecture of attention-based networks. The experimental results and analysis on different datasets demonstrate the effectiveness of our proposed framework.

There is some future work to be continued. For example, DARTS tends to learn some operations that need not learn. Phased training is a good method to avoid this problem. Furthermore, we will try to apply other NAS methods into our framework to observe their performance. Another interesting work is to explore our framework to search the architecture of other attention-based networks like transformer-based models.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[2] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2261–2269.

[3] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Conference on Neural Information Processing Systems*, 2015, pp. 2692–2700.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Conference on Neural Information Processing Systems*, 2017, pp. 5998–6008.

[5] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[6] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: stochastic neural architecture search," in *International Conference on Learning Representations*, 2019.

[7] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations*, 2017.

[8] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *International Conference on Machine Learning*, 2018, pp. 4092–4101.

[9] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8697–8710.

[10] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *European Conference on Computer Vision*, 2018, pp. 19–35.

[11] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI Conference on Artificial Intelligence*, 2019, pp. 4780–4789.

[12] M. Wistuba and T. Pedapati, "Inductive transfer for neural architecture optimization," *CoRR*, vol. abs/1903.03536, 2019.

[13] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *International Conference on Learning Representations*, 2019.

[14] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *International Conference on Learning Representations*, 2015.

[15] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International Conference on Machine Learning*, vol. 37, 2015, pp. 2048–2057.

[16] H. Larochelle, D. Erhan, A. C. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *International Conference on Machine Learning*, 2007, pp. 473–480.

[17] C. W. Hsu, C. C. Chang, and C. J. Lin, "A practical guide to support vector classication," 2010.

[18] J. Y. Hesterman, L. Caucci, M. A. Kupinski, H. H. Barrett, and L. R. Furenlid, "Maximum-likelihood estimation with a contracting-grid search algorithm," *IEEE Transactions on Nuclear Science*, vol. 57, no. 3, p. 1077, 2010.

[19] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, 2012.

[20] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing neural network architectures using reinforcement learning," in *International Conference on Learning Representations*, 2017.

[21] Z. Zhong, J. Yan, W. Wu, J. Shao, and C. Liu, "Practical block-wise neural network architecture generation," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2423–2432.

[22] L. Xie and A. L. Yuille, "Genetic CNN," in *IEEE International Conference on Computer Vision*, 2017, pp. 1388–1397.

[23] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *International Joint Conference on Artificial Intelligence*, 2018, pp. 5369–5373.

[24] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *International Conference on Machine Learning*, 2017, pp. 2902–2911.

[25] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *International Conference on Learning Representations*, 2018.

[26] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *International Conference on Learning Representations*, 2019.

[27] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Learning and Intelligent Optimization Conference*, ser. Lecture Notes in Computer Science, vol. 6683, 2011, pp. 507–523.

[28] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Conference on Neural Information Processing Systems*, 2012, pp. 2960–2968.

[29] J. Bergstra, D. Yamins, and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *International Conference on Machine Learning*, 2013, pp. 115–123.

[30] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian optimization of machine learning hyperparameters on large datasets," in *International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 54, 2017, pp. 528–536.

[31] S. Falkner, A. Klein, and F. Hutter, "BOHB: robust and efficient hyperparameter optimization at scale," in *International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, 2018, pp. 1436–1445.

[32] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *International Conference on Machine Learning*, vol. 37, 2015, pp. 2113–2122.

[33] S. Saxena and J. Verbeek, "Convolutional neural fabrics," in *Conference on Neural Information Processing Systems*, 2016, pp. 4053–4061.

[34] F. Pedregosa, "Hyperparameter optimization with approximate gradient," in *International Conference on Machine Learning*, vol. 48, 2016, pp. 737–746.

[35] K. Ahmed and L. Torresani, "Connectivity learning in multi-branch networks," *CoRR*, vol. abs/1709.09582, 2017.

[36] R. Shin, C. Packer, and D. Song, "Differentiable neural network architecture search," in *International Conference on Learning Representations*, 2018.

[37] A. Hundt, V. Jain, and G. D. Hager, "sharpdarts: Faster and more accurate differentiable architecture search," *CoRR*, vol. abs/1903.09900, 2019.

[38] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of english: The penn treebank," *Computational Linguistics*, vol. 19, pp. 313–330, 1993.

[39] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing LSTM language models," in *International Conference on Learning Representations*, 2018.

[40] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," in *International Conference on Learning Representations*, 2017.

[41] D. Liu, S. Chuang, and H. Lee, "Attention-based memory selection recurrent network for language modeling," *CoRR*, vol. abs/1611.08656, 2016.

[42] K. M. Tran, A. Bisazza, and C. Monz, "Recurrent memory networks for language modeling," in *Human Language Technology: Conference of the North American Chapter of the Association of Computational Linguistics*, 2016, pp. 321–331.

[43] J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber, "Recurrent highway networks," in *International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 70, 2017, pp. 4189–4198.

[44] G. Melis, C. Dyer, and P. Blunsom, "On the state of the art of evaluation in neural language models," in *International Conference on Learning Representations*, 2018.

[45] Z. Yang, Z. Dai, R. Salakhutdinov, and W. W. Cohen, "Breaking the softmax bottleneck: A high-rank RNN language model," in *International Conference on Learning Representations*, 2018.

[46] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *IEEE Conference on Computer Vision and Pattern Recognition*, June 2019.

[47] H. Inan, K. Khosravi, and R. Socher, "Tying word vectors and word classifiers: A loss framework for language modeling," in *International Conference on Learning Representations*, 2017.

[48] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1412–1421.

[49] R. Luo, F. Tian, T. Qin, E. Chen, and T. Liu, "Neural architecture optimization," in *Conference on Neural Information Processing Systems*, 2018, pp. 7827–7838.