# Recurrent Neural Architecture Search based on Randomness-Enhanced Tabu Algorithm

Kai Hu*, Shuo Tian*, Shasha Guo*, Nan Li*, Li Luo*, Lei Wang*
*College of Computer Science and Technology
*National University of Defense Technology
*Changsha, China
hukai18@nudt.edu.cn, leiwang@nudt.edu.cn

***Abstract*—Deep neural networks have achieved highly competitive performance in multiple tasks in recent years. However, discovering state-of-the-art neural network architectures requires substantial effort from human experts. To speed up the process, neural architecture search (NAS) has been proposed to search promising architectures automatically. Nevertheless, the search process of NAS is computing-expensive and time-consuming, which even costs thousands of GPU days. In this paper, to solve the bottleneck, we apply the randomness-enhanced tabu algorithm as a controller to sample candidate architectures, which balances the global exploration and local exploitation for the architectural solutions. In addition, more aggressive weight-sharing strategy is introduced into our method, which significantly reduces the overhead of evaluating sampled architectures. Our approach discovers the recurrent neural architecture within 0.78 GPU hour, which is 15.3x more efficient than ENAS [1] in terms of search time, and the architecture we discovered achieves the test perplexity of 56.1 on Penn Tree Bank (PTB) dataset, which is lower than ENAS by 2.2. In addition, we further demonstrate the usefulness of the learned architecture by transferring it to wiki-text-2 (WT2) dataset well. Moreover, the extended experiments on the WT2 dataset also show promising results.**

***Index Terms*—deep learning, neural architecture search, tabu algorithm, weight sharing**

## I. Introduction

Deep neural networks are efficient and flexible models that can perform many deep learning tasks, such as computer vision [2], social network filtering [3] and natural language processing [4]. However, designing a deep neural network architecture is time-consuming, labor-intensive and computing-intensive. Neural Network Search (NAS) technique alleviates this problem. Surprisingly, NAS often breaks through the limitations of human minds and achieves unexpected results [1], [5], [6], [7], [8]. The process of NAS is illustrated in Fig. 1.

NAS was first proposed by Zoph & Le [6]. In this work, an RNN controller is trained to sample a candidate architecture. The sampled architecture is trained on the training set, and then its performance is measured on the validation dataset. The controller uses the performance metric as a reward signal to update the controller network to find more promising

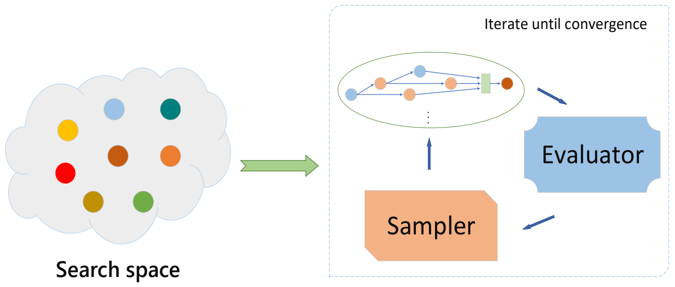

Fig. 1: The process of NAS. First, the search space is defined as a huge set containing plenty of architecture candidates. Next, a sampler is constructed by a certain strategy, such as reinforcement learning and evolutionary algorithm, to sample good network architectures. Afterwards, an evaluator evaluates the sampled architectures according to a certain approximate evaluation strategy to obtain performance metrics, such as accuracy, perplexity, etc. Finally, the metrics work as signals to update the evaluator. The process will iterate multiple times until convergence and then we will discover a promising architecture.

architectures. The architecture discovered in this way achieves state-of-the-art performance on an image classification task. However, obtaining the architecture required about 2000 GPU days.

ENAS [1] is notably 1000x less expensive than previous work. Similar to [6], ENAS also uses reinforcement learning [9] to train the LSTM [10] controller to sample candidate architectures. The core idea of ENAS is forcing all candidate architectures to share weights to avoid training each child model from scratch to convergence.

However, in both of these two implementations, the controller network needs to be trained well to sample a better candidate architecture. This is not only time-consuming but also increases the complexity of the search process.

The search space of ENAS is represented as a huge computational graph, and each candidate architecture is a subgraph of it. Consequently, ENAS could be viewed as an optimization problem of how to select a subgraph that maximizes the expected reward on a validation dataset. Inspired by this, in the

Kai Hu and Shuo Tian contribute to this article equally. This research is founded by the National Key R&D Program of China [grant numbers 2018YFB2202603] and in part by the National Natural Science Foundation of China [grant numbers 61802427] and [grant numbers 61832018].

paper, we propose the randomness-enhanced tabu algorithm as the controller to sample promising candidate architectures more efficiently. This reduces the time of sampling candidate architectures due to not needing to train a network as a controller. Furthermore, we improve the tabu algorithm [11] by introducing more randomness that balances the global exploration and local exploitation for the candidate solutions, which makes it easier to get high-quality architectures within less search time.

In addition, evaluating the quality of sampled architectures requires a lot of computing resources. The sampled architectures first are trained on the training dataset and then evaluated on the validation dataset before obtaining the performance metrics. The overhead will be unacceptable if the architectural performance metrics are not evaluated approximately. Therefore, we introduce weight sharing proposed by Pham et al. [1] into our method, which makes evaluation for candidate architectures faster.

In ENAS, sharing weights are trained for 150 epochs. However, the well-trained parameters are still approximate parameters for each candidate architecture so we can't obtain a real performance for each architecture only by using these well-trained parameters. Moreover, we find that paying more focus on network topology instead of network parameters is easier to obtain a better architecture. In this paper, more aggressively, we only train the sharing parameters for one epoch and further reduce the search time.

The main contributions of the paper follow as:

- It is the first time that a tabu algorithm is used in the sampling stage to speed up recurrent neural architecture search.
- We improve tabu algorithm by introducing randomness that balances the global exploration and local exploitation for recurrent candidate architectural solutions.
- We introduced weights sharing to our method. Furthermore, sharing weights are trained for only one epoch, which reduces search time without impacting performance.

On the Penn Tree Bank (PTB) [12] dataset, 30,000 architectures are searched using only 0.78 GPU hour, which is 15.3x more efficient than ENAS. In addition, the architecture we discovered achieves the test perplexity of 56.1,which is lower than ENAS by 2.2. We further demonstrate the usefulness of the learned architecture by transferring it to wiki-text-2 (WT2) dataset well. Eventually, 40,000 architectures are searched using 1.10 GPU hours on the WT2 dataset. And we obtain the final recurrent architecture, which outperforms the performance of the recurrent cells discovered by ENAS and our method on PTB dataset.

## II. Related Work

Zoph & Le [6] firstly presented modern algorithm automating architecture design, and resulting architectures can indeed outperform manually designed state-of-the-art neural networks. Similar to [6], some works [5], [13], [14] used reinforcement learning for neural architecture search, which formulate neural architecture search (NAS) as a graph search problem.

In addition to reinforcement learning (RL), evolutionary algorithm is also a mainstream search strategy. Real et al. [8] introduced evolutionary algorithm as a controller to sample candidate architectures. The controller retained good genotypes through the tournament selection mechanism and generated new genotypes through the mutation. In this process, no controller networks were trained. Therefore, compared with the RL method, evolutionary algorithm takes less time in the sampling stage.

Unlike conventional approaches which need a discrete search space, DARTS [7] is a novel gradient-based framework that enables neural architecture search to use gradient descent by introducing a continuous relaxation of the search space. However, DARTS obtained the recurrent architecture for PTB dataset requiring 1 GPU days.

In addition, some other optimization algorithms are used as controllers for NAS. PNAS [5] used sequential model-based optimization (SMBO) strategy to search for convolutional neural architectures in order of increasing complexity. Self-adaptive harmony search (SAHS) algorithm [15] was used to find the promising convolutional neural network architecture for image recognition tasks. The implementation was largely based on the existing network (eg.VGG16 [16] and ImageNet [17]). Besides, particle swarm optimization (PSO) algorithm [18] was used to evolve convolutional neural architectures. The core point is that learning a block on a smaller dataset and transferring the learned block to a larger dataset could reduce the computation cost successfully.

Unfortunately, aforementioned methods have a limited improvement for the performance and search time. And NAS problems are computing-intensive, which even cost hundreds or thousands of GPU days for discovering promising architectures. Surprisingly, weights sharing was proposed by Pham et al. [1], which made ENAS 1000x less expensive than standard Neural Architecture Search. The basic principle is that sharing parameters among child models allows ENAS to deliver strong empirical performances in the evaluation process.

However, Sciuto et al. [19] had put forward different views on parameter sharing. The claim is that, weight sharing negatively impacts the real ranking of candidate architectures. Besides, a controversial point of view was raised in this paper, the random policy even outperformed state-of-the-art NAS algorithms due to the limitations of the current search space.

In addition, Gaier et al. [20] proposed that network topology is more important than the weight parameters of neural networks for certain tasks. Extremely, the sharing weight parameter is even one number for all connections in the experiments, which makes the evaluation process for all the candidate architectures more efficient.

Inspired by these, in our approach, we pay more attention to random search and network topology instead of well-trained weight parameters.

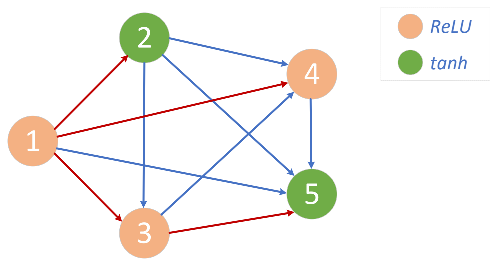

Fig. 2: Suppose our search space consists of 5 computational nodes, and the subgraph combined by the red arrows is an architecture we discovered. Here, node 1 is the input of the architecture, and nodes 2, 4 and 5 are the outputs of the architecture.

## III. Method

In this section, we propose a method to speed up recurrent neural architecture search. First, the reduced search space is applied to our method, which is based on observations of the previous works [1], [7] and specific analysis for two abandoned activation functions, namely $identity$ and $sigmoid$. Furthermore, we demonstrate the implementation details of the randomness-enhanced tabu algorithm. Ultimately, a more aggressive weight-sharing technique is introduced into our approach, which further accelerates the search process.

### A. Reduced Search Space

In terms of search space, we follow the method of ENAS and PNAS [5] mostly. As shown in Fig. 2, the graph represents the entire search space, and every candidate architecture can be defined by using a single directed acyclic graph (DAG), which is a subgraph of the complete graph.

In fact, we can represent an architecture with only two vectors, vector $pre\_nodes$ and vector $activation\_functions$. In particular, $pre\_nodes[i]$ refers to the previous-node which is connecting with node $i$ and $activation\_function[i]$ refers to the activation function which is applied to node $i$. For instance, all previous-nodes of node 5 consist of node 1, 2, 3 and 4. As shown in Fig. 2, node 3 is selected to connect to node 5 from them and $tanh$ is selected as the activation function of node 5. The process above can be represented with two formulas: $pre\_nodes[5] = 3$ and $activation\_function[5] = "tanh"$. Following this mode, we can represent any candidate architecture with a sequence of these two formulas.

Specifically, the recurrent cell has 12 nodes and we only allow 2 activation functions (namely $tanh, ReLU$) that differ from Pham's setup [1]. The reasons why we reduce the search space are explained below.

Firstly, in ENAS, we find that the LSTM controller never chose $identity$ and $sigmoid$ and DARTS [7] only chose one $identity$, and never chose $sigmoid$ in their final architectures. Therefore, we think that using two activation functions ($ReLU$ and $tanh$) is sufficient to obtain a promising architecture.

Secondly, activation functions are pivotal for neural networks to learn something non-linear complex functional mappings between the inputs and outputs. However, the $identity$ activation function does not satisfy this property. When $identity$ activation function is used in multiple layers of a neural network, the entire network is even equivalent to a single-layer model. Because $identity$ plays a crucial role to retain the complex information of the previous layer in a deep neural network. However, our recurrent cell follows ENAS and has only 12 nodes. The cell is simple and we prefer to introduce more non-linear properties. Therefore, $identity$ is not included in our search space.

In addition, we enhance the simple transformations between nodes in the constructed recurrent cell with highway connections [21] following ENAS. And highway connections technology is implemented with $sigmoid$. Thererfore, the non-linear property introduced by $sigmoid$ has always existed in our cell and consequently $sigmoid$ is also not included in our search space.

Based on aforementioned observations and analysis, the reduced search space is applied to our experiments. In addition, the size of search space is $a^N \times N!$ where $N$ is the number of nodes in the architecture represented by a DAG, and $a$ is the number of kinds of activation functions applied to the nodes. Indeed, compared to ENAS, the search space has been reduced by 4000x because we delete two activation functions, $identity$ and $sigmoid$. In this reduced search space, multiple rounds of experiments have demonstrated that we can obtain a promising recurrent architecture with less time.

### B. Randomness-Enhanced Tabu Algorithm

Tabu search is designed to manage a hill-climbing heuristic and could adapt to manage any neighborhood exploitation heuristic in discrete domains. The core of this algorithm is restricting the feasible neighborhood domain by neighbors that are excluded. Specifically, tabu algorithm avoids cyclic search by introducing the local neighborhood search mechanism and corresponding tabu mechanism. In addition, tabu algorithm can break the tabu states in some ways to avoid falling into a local optimum. In this paper, we improve the algorithm in two aspects, providing a good initial solution and enlarging the randomness. The randomness-enhanced tabu search method we used is summarized in Algorithm 1.

*1) Initial Solution of Tabu Algorithm:* An initial solution of tabu algorithm is often chosen randomly, but we will achieve a better solution with less time if a good initial solution is assigned to this algorithm. Here we search 100 random candidate architectures and then select the best one as the initial solution. The process is described by the function at line 4 of Algorithm 1.

*2) Enlarging the Randomness:* Tabu search is a local search algorithm, and new solutions are often limited around the current optimal solution. Therefore, the original tabu algorithm is easy to fall into local optimal results. Fig. 3 illustrates that the original tabu algorithm is always trapped in the local optimal solution around 5,000 epochs and keep the

**Algorithm 1** Randomness-Enhanced Tabu Algorithm

```
 1: Initialize: $tabu\_list \leftarrow \emptyset$, $tabu\_size$, $R, S$
 2: for $round = 1 \rightarrow R$ do
 3:     if $round == 1$ then
 4:         $model.arch$ = INITIALARCHGENERATOR()
 5:         $tabu\_list.append(model.arch)$
 6:     else
 7:         $model.arch$ = RANDOMSAMPLEARCH()
 8:     end if
 9:     for $step = 1 \rightarrow S$ do
10:         $new\_model.arch$ =
                NEWARCHGENERATOR($model.arch$)
11:         if $new\_model.arch \notin tabu\_list$ then
12:             $new\_model.ppl$ =
                    ARCHEVALUATE($new\_model.arch$)
13:         end if
14:         while $new\_model.ppl < Tbest$ do
15:             if $len(tabu\_list) < tabu\_size$ then
16:                 $tabu\_list.append(new\_model.arch)$
17:                 $Tbest = new\_model.ppl$
18:             end if
19:         end while
20:         $model.arch = tabu\_list[-1]$
21:     end for
22: end for
```

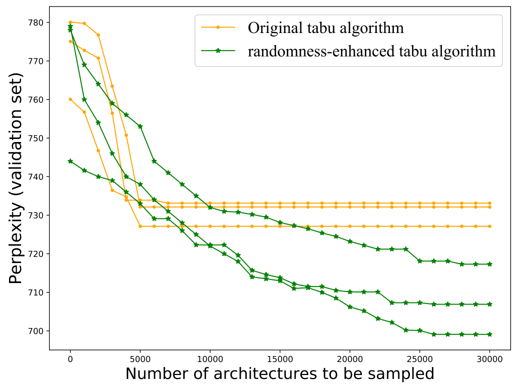

Fig. 3: Comparison of the original tabu algorithm and the randomness-enhanced tabu algorithm in the sampling process. X-axis: The number of architectures to be sampled; Y-axis: The perplexity of the sampled architecture evaluated on the validation set.

sub-optimal solution unitil the algorithm ends. However, the randomness-enhanced tabu algorithm constantly approaches a better solution until the algorithm ends.

We will explain randomness-enhanced tabu algorithm in detail below. Line 1 in Algorithm 1 represents that we initialize four parameters: $tabu\_list$, $tabu\_size$, $R$ and $S$. Specifically, $tabu\_list$ stores previously-visited prohibited solutions. The role of $tabu\_list$ is to prevent the searching stage from appearing in the loop. And $tabu\_size$ is just the size of $tabu\_list$. In the paper, we set it to 1000, which is big enough to store all the visited candidate solutions. Significantly, $R$ represents the total number of random solutions that we plan to generate. Line 4 reveals a function, which provides an good initial solution to the algorithm. Lines 2-7 represent the process in which we randomize an architecture as a new solution after every $S$ steps exploitation for the current architecture. Then this process will be iterated $R$ times, which means that a total of $R$ global random solutions will be generated during the entire search process.

Lines 9-19 in Algorithm 1 represent the implementation of the original tabu search algorithm. Tabu search uses a neighborhood search procedure to iteratively move from one current solution $x$ to another new neighborhood solution $x'$, a total of $S$ times. Line 10 represents that the algorithm generates a new random architectural solution by randomly changing an edge or an activation function of the current architecture as shown in Fig. 4. In lines 11 and 12, when the new solution we sample does not exist in $tabu\_list$, the solution will be evaluated and then we will get its performance metric. As shown in lines 14-17, if the metric is better than the current optimal architecture, the new solution is the best architectural solution. When the $tabu\_list$ is not full, the new solution will be appended in the $tabu\_list$.

The original tabu algorithm is a local search algorithm, exploiting new solutions around the current optimal solution constantly. However, randomness-enhanced tabu algorithm introduces more global random candidate solutions. Recently, researchers pay more attention to random search. Sciuto et al. [19] believe random policy outperforms state-of-the-art NAS algorithms. It is a very controversial point, but we realize the importance of random search. Thence, we propose to enlarge the proportion of random solutions to avoid getting trapped in a sub-optimal solution.

Randomness-enhanced tabu algorithm balances the global exploration and local exploitation for recurrent candidate architectural solutions, which makes it easier to achieve a promising architecture. Besides, the algorithm makes it more efficient to sample new architectures. In particular, compared to ENAS, we don't need to train an LSTM controller in each epoch to enhance its sampling ability, while we only change an edge or an activation function in our approach to optimize current architecture and then introduce a random architecture after every $S$ steps.

## C. Weight Sharing

Weight sharing is a smart method for approximately evaluating the performance of a candidate recurrent architecture. Inspired by transfer learning, Pham et al. [1] first proposed weight sharing in ENAS. The main idea of weight sharing is forcing all candidate recurrent architectures to share weight parameters and delivering strong empirical performances in the evaluation process. Experimentally, weight sharing overwhelmingly improves the efficiency of the evaluation and

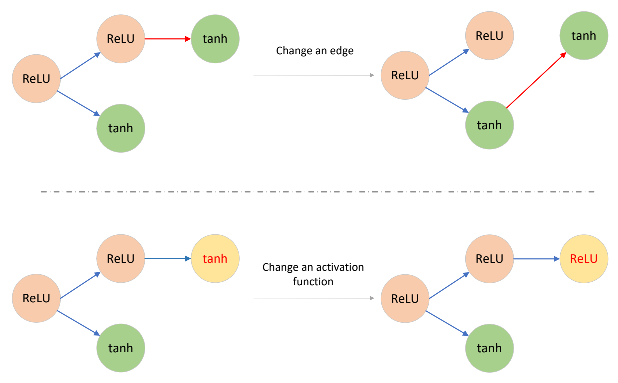

Fig. 4: Two ways to generate a new neighborhood architecture based on the current sampled architecture. Top: Change an edge by connecting the node with another previous-node. The red arrows show the change in the connection between edges; Bottom: Replacing an activation with another one. The yellow nodes indicate the change in the type of activation functions.

ENAS is 1000x less expensive than standard Neural Architecture Search.

To further speed up the search process, more aggressively, we train the sharing weights for only one epoch in our experiments. We will explain why we do this below.

Sciuto et al. [19] proposed that weight sharing negatively impacts the real ranking of candidate architectures. As described in [19], the sharing weights in ENAS trained for 150 epochs are still approximate parameters for each candidate architecture and we can't obtain a real ranking for all evaluated architectures with the sharing weights. In addition, training sharing weights for 150 epochs consumes a lot of time. Hence, we train the sharing weights for only one epoch. Intuitively, underfitting network weights are detrimental for evaluating an architecture. However, the underfitting sharing weights make the architecture search process more focused on the network topology as described in [20]. Experimentally, we find that the architecture discovered by our method using the underfitting sharing weights performs better than the architecture of ENAS in terms of performance. To sum up, the sharing weights trained for only one epoch encourages searching for the architectures with better network topologies.

In addition, we further shorten the architecture search time, which is 15.3x more efficient than ENAS. And in this way, we achieved a state-of-the-art recurrent cell, which is lower than the architecture discovered by ENAS by 2.2 for test perplexity on the PTB dataset.

## IV. Experiments and Results

### A. Experimental Setup

**Experimental details:** We apply our method to a language modeling task with Penn Treebank (PTB) and Wiki-Text2 (WT2) dataset. All the experiments are implemented in PyTorch and we train the resulting RNNs on a single GPU of NVIDIA 1080Ti. We use and modify the source code at [22] in our experiments. The sharing weights of all models are trained using stochastic gradient descent (SGD) with a learning rate of 20.0. In the training stage, the learning rate is 20.0 and decays by a factor of 0.96 per epoch starting at epoch 15. In addition, batch size is 64 in the training stage, while batch size is 1 in the validation stage.

**Baseline:** The main baseline is ENAS in our experiments. Hence, all the recurrent cells consist of 12 nodes following the setting of ENAS. For a fair comparison, we not only compare the original architecture in the ENAS paper, but also compare the architecture discovered in the reduced search space by using ENAS code.

Besides, we compare against random search in the reduced search space. We ensure identical conditions for random search (RS) and our method. Both of methods use the same computer code for network training and evaluation. Specifically, in the evaluation process, we introduce weight sharing to shorten search time like ENAS. However, we train the sharing weights for only one epoch in RS and our method in contrast to 150 epochs in ENAS. RS samples over 30,000 architecture models, which is equal to the number of models our method samples. It is more fair than ENAS and DARTS [7]. In particular, ENAS reports the results of only a single random architecture, and DARTS reports an architecture selected among 8 randomly sampled ones as the most promising one after training for 300 epochs only.

### B. Exploration of $R$ and $S$

As shown in Algorithm 1, we randomize an architecture as a new solution after every $S$ steps exploitation for the current architecture. And we plan to generate a total of $R$ global random solutions during the entire search process. Therefore, the total number of architectures we sampled over is $R \times S$. When the total number of sampled architectures is constant, the larger $R$ is, the better global solution space could be explored. In addition, the larger $S$ is, the better local solution space could be exploited.

The choice of parameters $R$ and $S$ in the experiment on PTB dataset is determined after 5 attempts. Firstly, we determine a total of 30,000 candidate architectures that will be sampled in the entire search process, which is same as ENAS. This means that $R \times S = 30,000$, and then we construct five combinations of parameters $R$ and $S$ in the order of increasing $R$. The five combinations are $(R = 300, S = 100)$, $(R = 600, S = 50)$, $(R = 1000, S = 30)$, $(R = 2000, S = 150)$, and $(R = 3000, S = 10)$ respectively. We train the final architecture discovered in each combination for only 50 epochs due to constrained time and computing resources. As illustrated in Fig. 5, the architecture discovered in the combination of $(R = 1000, S = 30)$ shows more promising performance than other combinations and achieves the minimum perplexity of 75.43 within 50 epochs. Therefore, $R = 1000$ and $S = 30$ are determined as configuration parameters in our experiments. That means our approach exploits 30 neighborhood solutions

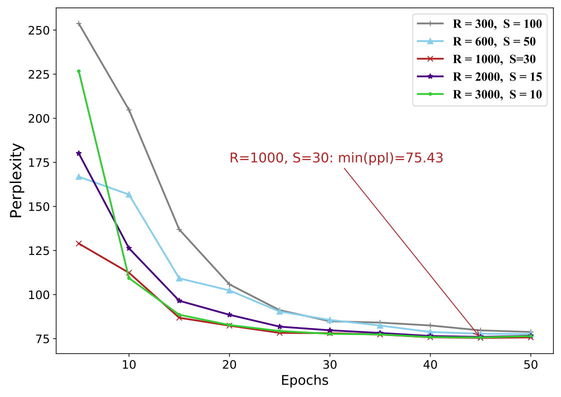

Fig. 5: The choice of $R$ and $S$ on PTB dataset. X-axis: The number of epochs; Y-axis: The test perplexity of the recurrent architectures with different configurations. During 50 epochs, the minimum value of test perplexity (75.43) appears in the fourth case: $R = 1000, S = 30$.

for a current candidate solution and then samples a random architecture for global exploration.

## C. Searching for Recurrent Architecture on PTB dataset

Our set of available operations includes $tanh$ and $ReLU$. while all the recurrent cell consists of $N = 12$ nodes. The very first intermediate node is obtained by two input nodes, and the rest of the nodes have only one input. In addition, all the nodes only have an output and the output is defined as the average of all the leaf nodes. Our recurrent network architeccture consists of only a single cell, which is same as ENAS. However, our method trains the sharing weights for only one epoch and searchs candidate architectures in a reduced search space, which only includes $tanh$ and $ReLU$ as activation functions. Similar to ENAS, each operation is enhanced with a highway bypass [21]. In addition, we enable batch normalization [23] in each node to prevent gradient explosion during architecture search. Fig. 6 illustrates the final recurrent architecture discovered by our method.

After exploring experiments on parameters $R$ and $S$, $R = 1000$ and $S = 30$ is determined as configuration parameters in the experiments of the PTB dataset. To enable comparable to ENAS, we limit the size of parameters to 24M. Table I presents the experimental results of different recurrent architectures on PTB dataset. We find that ENAS exceeds random search in terms of performance, but it takes much more time. Because random search trains sharing weights for only one epoch like our method, which accelerates the search process. In addition, random search is more efficient in sampling a new architecture than ENAS due to not needing to train an LSTM network as a sampler. However, random search sampling process is completely random, while the NAS algorithm, like ENAS and our method, will seek better solutions based on the corresponding rules constantly. Therefore, random search is

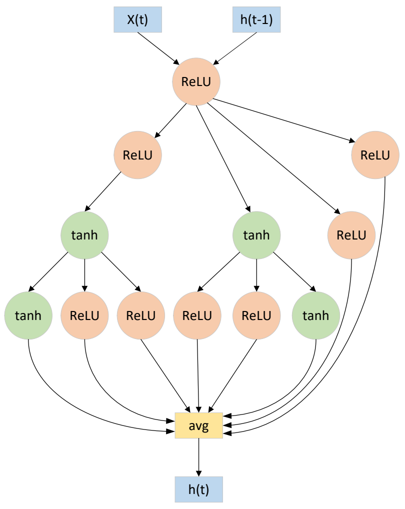

Fig. 6: The RNN architecture discovered by our method for PTB dataset.

TABLE I: Comparison with state-of-the-art language models on PTB dataset.

| Architecture | Perplexity |  | Params (M) | GPU time |
|---|---|---|---|---|
|  | valid | test |  |  |
| **Variational RHN [21]** | 67.9 | 65.4 | 23 | - |
| **LSTM [24]** | 60.7 | 58.8 | 24 | - |
| **LSTM+skip connections [25]** | 60.9 | 58.3 | 24 | - |
| **LSTM+5 softmax experts [26]** | - | 57.4 | - | - |
| **LSTM+15 softmax experts [26]** | 58.1 | 56.0 | 22 | - |
| **Random search** | 62.0 | 60.3 | 24 | 0.78 hour |
| **NAS [6]** | - | 63.7 | 25 | $10^4$ CPU days |
| **ENAS [1]** [1] | 60.3 | 58.3 | 24 | 12 hours |
| **ENAS [1]** [2] | 60.2 | 58.1 | 24 | 12 hours |
| **Our Method** | 58.1 | 56.1 | 24 | 0.78 hour |

[1] Obtained by training the corresponding architecture using our setup. The corresponding architecture is publicly released by the authors in the paper.
[2] The architecture is discovered in our reduced search space on PTB dataset using ENAS code and then is trained using our setup.

not as good as the NAS algorithm when the number of samples is not large enough.

In addition, Table I presents that random search has a better performance than pure NAS. We will explain it from the following two aspects. Firstly, the reduced search space is applied in random search. Searching in the reduced search space is easier to get a promising architecture. Besides, only 15,000 architectures are evaluated because pure NAS is extremely time-consuming, while random search search for 30,000 architectures like ENAS and our method.

The recurrent cell discovered by our method achieves the test perplexity of 56.1, which is on par with the state-of-the-art recurrent model enhanced by a mixture of softmaxes [26]. And our recurrent cell outperforms all the rest of recurrent architectures which are either manually designed or automatically searched by NAS algorithms. ENAS reports the test perplexity of 55.8 in the paper. However, the result was obtained on

TABLE II: Comparison with state-of-the-art language models on WT2 dataset.

| Architecture | Perplexity | | Params (M) | GPU time |
|---|---|---|---|---|
| | valid | test | | |
| **LSTM+augmented loss [27]** | 91.5 | 87.0 | 28 | - |
| **LSTM+continuous cache pointer [28]** | - | 68.9 | - | - |
| **LSTM [24]** | 69.1 | 66.0 | 33 | - |
| **LSTM+skip connections [25]** | 69.1 | 65.9 | 24 | - |
| **LSTM+15 softmax experts [26]** | 66.0 | 63.3 | 33 | - |
| **ENAS [1]** [1] | 71.9 | 70.0 | 33 | 12 hours |
| **ENAS [1]** [2] | 70.6 | 69.1 | 33 | 12 hours |
| **Our Method (searched on PTB)** | 69.4 | 67.5 | 33 | 0.78 hour |
| **Our Method (searched on WT2)** | 69.1 | 67.3 | 33 | 1.10 hours |

[1] Transfering the architecture searched on PTB dataset to WT2 dataset. The architecture is publicly released by the authors in the paper.
[2] The architecture is discovered in our reduced search space on WT2 dataset using ENAS code and then is trained using our setup.

TensorFlow platform instead of PyTorch platform, and ENAS used Tensor Processing Unit (TPU) to train the resulting recurrent cell. In our platform and experimental environment, the resulting recurrent cell discovered by ENAS achieved the test perplexity of 58.3, which is not as good as our recurrent cell. In addition, for a fair comparison, we use ENAS code to search a recurrent architecture in our reduced search space. The result shows that the performance of the architecture searched in the reduced search space slightly outperforms the performance of the original architecture of ENAS. This demonstrates that searching in the reduced search space is easier to obtain a promising architecture.

In terms of efficiency, the overall cost is 47 minutes on a single GPU of NVIDIA 1080Ti, which is significantly 15.3x faster than ENAS.

There is two main reasons why our method is more efficient than ENAS. Firstly, we train the sharing weights for only one epoch in contrast to 150 epochs in ENAS. Secondly, ENAS has to train the LTSM controller during the process of architecture search, while our method only needs to change an edge or an activation function in every epoch.

## D. Transfer to WT2 dataset

WT2 is a larger dataset than PTB. To validate the usefulness of our recurrent cell learned on PTB dataset, we transfer the architecture learned on PTB dataset to WT2 dataset. In addition, we limit the size of parameters to 33M for all models in consideration of increasing complexity. We train the recurrent cells discovered by ENAS and our method with the identical experimental environment.

Table II presents the results for all the recurrent cells on WT2 dataset. The recurrent cell we discovered achieves the test perplexity of 67.5, which is lower than ENAS by 0.5. It demonstrates that the recurrent architecture discovered by our method transfers to WT2 dataset better than ENAS. In addition to greatly enhanced efficiency, the experimental results further illustrates the superiority of our approach, that the architecture we discovered is scalable.

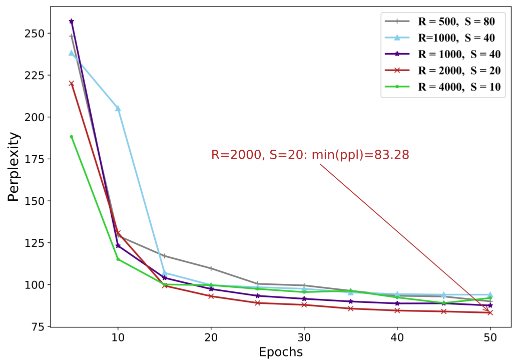

Fig. 7: The choice of $R$ and $S$ on WT2 dataset. X-axis: The number of epochs; Y-axis: The test perplexity of the recurrent architectures with different configurations. During 50 epochs, the minimum value of test perplexity (83.28) appears in the fourth case: $R = 2000, S = 20$.

## E. Searching for Recurrent Architecture on WT2 dataset

We further search for the recurrent cell on WT2 dataset. Our set of available operations still includes $tanh$ and $ReLU$ and our recurrent cell consists of $N = 12$ nodes. Because the dataset is larger, we search 40,000 recurrent architectures.

First, we explore the combination of $R$ and $S$. And we train the final architecture discovered in each combination for only 50 epochs. The total number of recurrent architectures we sampled is $R \times S = 40,000$. Five combinations are determined in the order of increasing $R$, which is $(R = 500, S = 80)$, $(R = 800, S = 50)$, $(R = 1000, S = 40)$, $(R = 2000, S = 20)$, and $(R = 4000, S = 10)$ respectively. As illustrated in Fig. 7, $R = 2000$ and $S = 20$ is the best choice. With this configuration, the architecture we discovered achieves the minimum test perplexity of 83.28 within 50 epochs. It means that our method exploits 20 neighborhood solutions for the current candidate architecture and then samples a random solution for global exploration.

With the configuration of $R = 2000, S = 20$, we discover the final recurrent cell as shown in Fig. 8. As shown in Table II, the recurrent cell discovered by our method achieves the test perplexity of 67.3.

In terms of ENAS, we use the original ENAS code to search for the architecture in the reduced search space. The results show that the performance of the architecture searched in the reduced search space on the WT2 dataset is better than the performance of the architecture transferred from PTB dataset.

However, the overall results on the WT2 dataset are less strong than those on the PTB dataset in Table I. Several models designed by human experts [24], [25], [26] exceed our recurrent architecture. Our architecture is an RNN, which is naturally weaker than LSTM for processing language tasks. Besides, these models optimized original LSTM by using some techniques, such as increasing limitation of $Softmax\ bottleneck$.

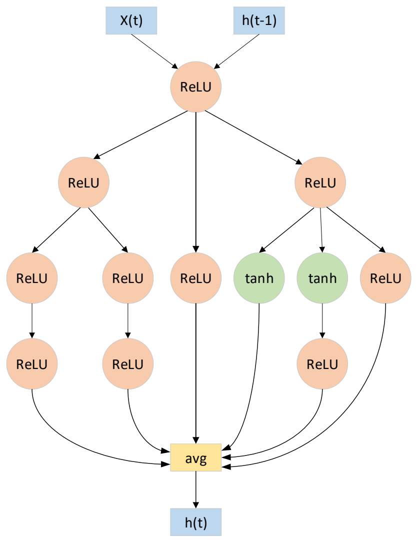

Fig. 8: The RNN architecture discovered by our method for WT2 dataset.

In terms of efficiency, the overall cost is 66 minutes on a single GPU of NVIDIA 1080Ti to search the architecture on WT2 dataset.

## V. Conclusion

Under the premise of guaranteeing the high quality of resulting architectures, our priority is enhancing the efficiency of NAS. In this paper, we propose that randomness-enhanced tabu algorithm works as a controller to sample recurrent architectures, which could balance the global exploration and local exploitation of solutions. In addition, reduced search space is applied to our method, which has a positive effect on searching for a promising neural architecture. In the experiments, we found a state-of-the-art recurrent architecture on PTB dataset and the architecture is scalable for WT2 dataset. In addition, the extended experiments on the WT2 dataset also show promising results. In the process of evaluating the sampled architectures, weight sharing plays a central role to significantly shorten search time. More aggressively, we only train the shared parameters for one epoch to pay more focus on network topology instead of network parameters. In the future, we will explore more possibilities for search space and assessment strategies to address more complex NAS tasks.

## References

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

[2] D. CireşAn, U. Meier, J. Masci, and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification," *Neural networks*, vol. 32, pp. 333–338, 2012.

[3] D. T. Nguyen, F. Alam, F. Ofli, and M. Imran, "Automatic image filtering on social networks using deep learning and perceptual hashing during crises," *arXiv preprint arXiv:1704.02602*, 2017.

[4] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[5] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 19–34.

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

[7] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[8] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4780–4789.

[9] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[10] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[11] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & operations research*, vol. 13, no. 5, pp. 533–549, 1986.

[12] M. Marcus, G. Kim, M. A. Marcinkiewicz, R. MacIntyre, A. Bies, M. Ferguson, K. Katz, and B. Schasberger, "The penn treebank: annotating predicate argument structure," in *Proceedings of the workshop on Human Language Technology*. Association for Computational Linguistics, 1994, pp. 114–119.

[13] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2017, pp. 497–504.

[14] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[15] Y.-F. Huang and J.-S. Liu, "Optimizing convolutional neural network architecture using a self-adaptive harmony search algorithm."

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[18] B. Wang, B. Xue, and M. Zhang, "Particle swarm optimisation for evolving deep neural networks for image classification by evolving and stacking transferable blocks," *arXiv preprint arXiv:1907.12659*, 2019.

[19] C. Sciuto, K. Yu, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," *arXiv preprint arXiv:1902.08142*, 2019.

[20] A. Gaier and D. Ha, "Weight agnostic neural networks," 2019.

[21] J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber, "Recurrent highway networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 4189–4198.

[22] T. Kim, "Enas-pytorch," https://github.com/carpedm20/ENAS-pytorch , 2018.

[23] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[24] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing lstm language models," *arXiv preprint arXiv:1708.02182*, 2017.

[25] G. Melis, C. Dyer, and P. Blunsom, "On the state of the art of evaluation in neural language models," in *arXiv:1707.05589*, 2017.

[26] Y. Zhilin, D. Zihang, S. Ruslan, and C. William, "Breaking the softmax bottleneck: A high rank rnn language model," in *ICLR*, 2018.

[27] I. Hakan, K. Khashayar, and R. Socher, "Tying word vectors and word classifiers: a loss framework for language modeling," in *ICLR*, 2017.

[28] G. Edouard, J. Armand, and N. Usunier, "Improving neural language models with a continuous cache," in *ICLR*, 2017.