RankNAS: Efficient Neural Architecture Search by Pairwise Ranking

Chi Hu, Chenglong Wang, Xiangnan Ma, Xia Meng, Yinqiao Li, Tong Xiao, Jingbo Zhu, Changliang Li

1NLP Lab, School of Computer Science and Engineering
Northeastern University, Shenyang, China
2NiuTrans Research, Shenyang, China
3Kingsoft AI Lab, Beijing, China

huchinlp@gmail.com, clwang1119@gmail.com, {xiaotong,zhujingbo}@mail.neu.edu.cn

Abstract

This paper addresses the efficiency challenge of Neural Architecture Search (NAS) by formulating the task as a ranking problem. Previous methods require numerous training examples to estimate the accurate performance of architectures, although the actual goal is to find the distinction between “good” and “bad” candidates. Here we do not resort to performance predictors. Instead, we propose a performance ranking method (RankNAS) via pairwise ranking. It enables efficient architecture search using much fewer training examples. Moreover, we develop an architecture selection method to prune the search space and concentrate on more promising candidates. Extensive experiments on machine translation and language modeling tasks show that RankNAS can design high-performance architectures while being orders of magnitude faster than state-of-the-art NAS systems.

1 Introduction

Neural Architecture Search (NAS) has advanced state-of-the-art on various tasks, such as image classification (Zoph et al., 2018; Pham et al., 2018; Real et al., 2019; Tan et al., 2019), machine translation (Fan et al., 2020; So et al., 2019), and language modeling (Pham et al., 2018; Liu et al., 2019; Jiang et al., 2019; Li et al., 2020). Despite the remarkable results, conventional NAS methods are computationally expensive, requiring training millions of architectures during search. For instance, obtaining a state-of-the-art machine translation model with an evolutionary algorithm requires more than 250 GPU years (So et al., 2019).

Several techniques have been proposed to improve the search efficiency, such as sharing parameters among all architectures (Pham et al., 2018; Cai et al., 2018; Zhong et al., 2018), predicting the performance instead of full training (Liu et al., 2018; Baker et al., 2018; Wen et al., 2020; Wei et al., 2020), and searching over a continuous space (Liu et al., 2019; Jiang et al., 2019; Li et al., 2020). Unfortunately, these approaches still suffer from the high cost of predicting the performance of each candidate architecture. An inherent reason for this is that obtaining accurate performance requires training numerous neural networks to convergence, as described in Sec. 2.2. However, it is unnecessary to predict the model performance as in previous NAS methods. Rather, all we need is to distinguish architectures of different quality in NAS, say, ranking these architectures.

In this paper, we approach the problem by formulating NAS as a ranking task. Here we propose RankNAS, a ranking model for comparing different architectures. One of the key challenges is that directly ranking all architectures in a large search space is still computationally infeasible. Therefore, we adopt the pairwise method (Burgess et al., 2005; Bau et al., 2015), where the ranking problem is reduced to a binary classification problem over architecture pairs. To speed up RankNAS further, we develop an architecture selection method that chooses the most promising architectures for evalu-
ation according to the importance of features, e.g., the topology of architectures.

We test RankNAS on well-established machine translation and language modeling benchmarks. Experiments show that RankNAS is orders of magnitude faster than standard NAS systems and can find better architectures. Notably, RankNAS is generic to different tasks and evaluation metrics. It achieves competitive results on hardware-aware NAS tasks and is $10 \times$ faster than the HAT baseline (Wang et al., 2020). It also discovers new architectures that outperform vanilla Transformer by +1.8 BLEU points on the IWSLT’14 De-En data and +1.5 BLEU points on the WMT’14 En-De data, surpassing the Evolved Transformer (So et al., 2019) with 150,000× less search cost.

2 Preliminaries

NAS generally consists of two steps: 1) sample architectures from the pre-defined search space, and 2) estimate the performance of these samples. This work focuses on the performance estimation step, which is the efficiency bottleneck of NAS.

2.1 Search Space

The search space contains all possible architectures for the search. In this work, we take the Transformer architecture for description, but the discussed problem and solutions are general and can be applied to other models. Following HAT (Wang et al., 2020), we represent a Transformer architecture as a set of features and search for the optimal model configuration.

An overview of the search space is shown in Figure 2. It is extended from the HAT’s space and inspired by manually designed Transformer variants, including Relative Position Representations (Shaw et al., 2018) and Deep Transformer (Wang et al., 2019). The search space can also be represented as a supernet where each sub-network is a unique architecture. The search space contains around $10^{23}$ possible architectures, as detailed in Appendix A.1. It is computationally prohibited to explore such a large space with an exhaustive method.

2.2 Performance Estimation

Let $\mathcal{A}$ denotes the search space, and each architecture in it is represented by a feature vector $\alpha$. Formally, the goal of NAS is to find the optimal architecture $\alpha^*$ with the best performance. The performance can be measured by some metrics, such as accuracy or latency. The performance estimation process consists of two steps: 1) estimate the performance of all architectures, and 2) choose the architecture with the optimal performance.

Without loss of generality, we define $\mathcal{S}(\cdot)$ as the performance evaluated by some metrics. The task here is to find the most promising architecture with maximum $\mathcal{S}(\cdot)$. Standard NAS methods solve this problem by learning to estimate the performance of each architecture. The objective is given by:

$$
\alpha^* = \arg\max_{\alpha} \mathcal{S}_{val}(w^*, \alpha) \\
\text{s.t. } w^* = \arg\max_{w} \mathcal{S}_{train}(w, \alpha)
$$

where $w$ is the weights associated with the architecture. $\mathcal{S}_{val}$ and $\mathcal{S}_{train}$ are the evaluation results on the validation set and training set, respectively.

Optimizing Eq. 1 is time-consuming as obtaining the optimal weights for each architecture requires training them to converge. Although we can share the weights among all architectures to
amortize the cost, performance evaluation is still nontrivial and requires numerous training steps.

3 NAS as Ranking

As mentioned in Sec. 2.2, the goal of NAS is to find promising architectures that achieve high performance on unseen data. NAS requires distinguishing whether the architectures are “good” or “bad” rather than predicting accurate performance. Therefore, it is natural to treat NAS as a ranking problem, in which the explicit goal is to rank different architectures correctly.

3.1 Pairwise Ranking

Problem Formulation. Given an architecture $\alpha$, we define a score $s$ on it by a function $r(\cdot)$:

$$s = r(\alpha, p)$$  \hspace{1cm} (2)

where $p$ is the parameter of the scoring function. We implement the scoring function with a gradient boosting decision tree, as detailed in Sec. 4.1.

We want to optimize $p$ such that $s$ assigns high scores to good architectures and low scores to bad architectures. This induces a ranking of the candidate architectures in the search space. It is infeasible to sort all candidate architectures in a large search space directly. A solution is to reduce the listwise ranking problem to the pairwise ranking problem. Fortunately, the properties of the NAS task allow us to achieve the goal. As described in Dudziak et al. (2020), the relation between any pair of performance is antisymmetric, transitive and connex. This makes it possible to rank all architectures via pairwise comparisons, substantially reducing the training complexity.

Training Set Construction. In pairwise ranking, the learning task is framed as a binary classification of architecture pairs into two categories: correctly ordered and incorrectly ordered. Given an architecture pair $(\alpha_i, \alpha_j)$ and the order of performance $P_{ij}$, we can construct training examples $(\alpha_i, \alpha_j, P_{ij})$ for the classification by comparing the two values. Note that $P_{ij}$ is a 0-1 variable. For example, if $\alpha_i$ is better than $\alpha_j$, we would add $(\alpha_i, \alpha_j, 1)$ and $(\alpha_j, \alpha_i, 0)$ to the training set.

Optimization. Consider a pair of architectures $(\alpha_i, \alpha_j)$, scored by $s_i$ and $s_j$, respectively. The probability of $\alpha_i$ being better than $\alpha_j$ is given by the difference through an activation function $g$:

$$P_{ij} = g(s_i - s_j)$$  \hspace{1cm} (3)

We assume that $P_{ij} \geq 0.5$ means $\alpha_i$ is better than $\alpha_j$ while $P_{ij} < 0.5$ means $\alpha_j$ is better than $\alpha_i$. Here we use a logistic function to achieve this goal:

$$P_{ij} = \frac{1}{1 + e^{-(s_i - s_j)}}$$  \hspace{1cm} (4)

Similarly, $P_{ji}$ can be induced by:

$$P_{ji} = \frac{1}{1 + e^{-(s_j - s_i)}} = 1 - P_{ij}$$  \hspace{1cm} (5)

Denote the gold score of $\alpha_i$ being better than $\alpha_j$ as $P_{ij}$. We use the cross-entropy loss function for the classification. The loss for a pair of inputs is:

$$L_{ij} = -(P_{ij} \log P_{ij} + P_{ji} \log P_{ji})$$
$$= (1 - P_{ij}) \cdot (s_i - s_j) + \log(1 + e^{-(s_i - s_j)})$$  \hspace{1cm} (6)

Compared with Eq. 1, Eq. 6 just requires $P_{ij}$. In particular, we use the intermediate performance measured on the validation set during training. It is much easier than assessing the accurate performance of candidate architectures. In this sense, the ranking model is “easier” to learn and may not need many training samples as in performance prediction. RankNAS also enables efficient optimization through gradient methods. Algorithm 1 describes the complete training process of the ranking model.

3.2 Applying Pairwise Ranking

Although the training time of the ranking model is heavily reduced, it is still challenging to apply it to the ranking of all architectures in the search space $A$. The challenge is that exploring all architectures is computationally expensive, even when the task is a binary classification.

Correlations between Features and Performance. We start by analyzing the effect of architectural features on estimated performance. Figure 4 illustrates the impact of the FFN dimension
on latency and the validation loss on the machine translation task. We observe that: (a) different architectural features have very different correlations with the same evaluation metric, and (b) the same features also have different influences on different metrics. For example, the latency monotonically increases when scaling the FFN dimension on CPUs, while it is almost unchanged on GPUs. Hence, it is natural to improve search efficiency by eliminating unimportant features.

**Feature Importance.** Inspired by previous feature selection methods (Breiman, 2001; Fisher et al., 2019), we measure the importance of an architectural feature (e.g., the number of layers) by calculating the increase in the model error after permuting the feature.

We assume that each architecture \( \alpha \) is represented by a feature vector \( f \in \mathcal{R}^{M \times N} \), where \( M \) is the number of different features, and \( N \) is the dimension of feature vectors. Also, we assume a set \( C \) that contains \( n \) architectures sampled from the search space. We first estimate the original model error \( L_{\text{total}} \) on \( C \) using the accumulation of the prediction errors. For any feature \( f_{i} \in f \), we randomize it for each architecture in \( C \). Then the randomized architectural features are passed to the ranking model and yield an error \( L_{i} \). The importance of the \( i \)-th feature \( f_{i} \) is defined by:

\[
I(f_{i}) = \frac{L_{i}}{L_{\text{total}}} \tag{7}
\]

where a higher value implies \( f_{i} \) is more important.

**Search Space Pruning.** It is easy to select valuable architectural features with the above measure. Given all features \( f \in \mathcal{R}^{M \times N} \), we discard those with a score less than a threshold \( \theta \) and obtain the selected features \( f' \in \mathcal{R}^{M' \times N} \), where \( M' < M \). Then we can prune the search space according to the selected features. For instance, if the feature Embedding Dimension is not selected, we will keep it fixed during the search. Finally, we only search over the architectures in the reduced search space.

An overview of the search process is presented in Figure 3. As described in Sec. 3.1, the training of the proposed ranking model is much cheaper than previous methods, which need to optimize the parameters for all architectures. Pruning search space further reduces the number of architectures to be evaluated. Also, the sampling procedure can be implemented with any existing NAS search strategy, e.g., Random Search (RS) or Evolution Algorithm (EA).

## 4 Experiments

### 4.1 Experimental Setups

We evaluate our methods on language modeling and machine translation tasks. In the experiments, we search for hardware-aware architectures and high-accuracy architectures.

**Training Setups.** For machine translation, we experiment on the IWSLT’14 De-En and WMT’14 En-De tasks using the identical settings as Wang et al. (2020). For language modeling, we experiment on the WikiText-103 dataset (Merity et al., 2019), we measure the importance of an architectural feature, 2) sample \( n \) architectures from the reduced search space by specific strategies, and 3) rank them with the trained ranking model and choose the best one. Here different color means different features.

**Figure 3:** The proposed search process consists of three steps: 1) prune the search space according to the importance of architectural features, 2) sample \( n \) architectures from the reduced search space by specific strategies, and 3) rank them with the trained ranking model and choose the best one. Here different color means different features.

**Figure 4:** The impact of FFN dimension on latency and validation loss. All results are obtained on the WMT’14 En-De task with the same settings described in Sec. 4.2.
| Hardware | Task | Method     | Latency (ms) | #Params | FLOPs (G) | BLEU | Search Cost (hours) |
|----------|------|------------|--------------|---------|----------|------|---------------------|
| Intel Xeon Silver 4114 CPU | WMT | Transformer  | 1031.4 | 213.0M | 12.7 | 28.4 | -                  |
|         |      | HAT        | 396.8 | 67.9M  | 4.2  | 28.5 | 335.1              |
|         |      | RankNAS    | **384.2** | 68.1M | **4.0** | **28.6** | **31.8**         |
|         | IWSLT| Transformer | 353.5 | 34.9M  | 1.6  | 34.4 | -                  |
|         |      | HAT        | **190.5** | **27.9M** | **1.4** | 34.5 | 31.7              |
|         |      | RankNAS    | 197.4 | 29.6M  | 1.5  | **34.6** | 7.2               |
| NVIDIA GTX 1080Ti GPU | WMT | Transformer | 249.6 | 213.0M | 12.7 | 28.4 | -                  |
|         |      | HAT        | 214.8 | 66.2M  | 4.1  | **28.5** | 302.1             |
|         |      | RankNAS    | **201.7** | **62.1M** | **3.9** | 28.4 | **30.2**         |
|         | IWSLT| Transformer | 200.9 | 34.9M  | 1.6  | 34.4 | -                  |
|         |      | HAT        | 159.4 | **33.9M** | 1.6  | 34.7 | 24.5              |
|         |      | RankNAS    | **148.2** | 35.4M | **1.4** | **34.7** | 5.8               |

Table 1: Comparisons of latency, model size, FLOPs, BLEU, and the overall search cost on machine translation tasks for the standard Transformer, HAT, and discovered architectures by our method. We mark the best results in bold for all metrics. Search costs are measured on a single RTX 2080Ti GPU.

2017) with the same settings as Baevski and Auli (2019). We set the maximum number of tokens per sample to 1,843 to fit the memory constraints and apply gradient accumulation to keep the same batch size as Baevski and Auli (2019)’s work. All models are trained with mixed precision on 8 NVIDIA RTX 2080 Ti GPUs except for IWSLT ones, which only take one GPU for training.

### Ranking Model Setups
We implement the ranking model (binary classifier) described in Sec.3.1 with LightGBM (Ke et al., 2017) and set the learning rate to 0.1. To prevent overfitting, we set the maximum number of leaves to 30 and the tree’s maximum depth to 6. We also use the default regularization terms and apply the early stopping strategy to the training. Specifically, the training stops if the validation score does not increase for 5 rounds. After training the ranking model, we apply the search space pruning method to find the most valuable architectural features for different tasks and hardware. There are two hyper-parameters for pruning: the sample size and the threshold. We set them to 200/1.15 and 300/1.25 for the hardware-aware architecture search and high-accuracy architecture search, respectively.

### Architecture Search Setups
Table 5 and Table 6 presents the search space of high-accuracy search for the translation tasks. We refer the readers to Wang et al. (2020)’s work for more details about the search space of hardware-aware architecture search. RankNAS is not restricted to a specific search strategy. We compare different search strategies in the experiments, including Random Search (RS) and Evolutionary Algorithm (EA). We apply uniform sampling for RS and use the same settings as Wang et al. (2020)’s work for EA. More specifically, the random search process will stop if the best-so-far architecture does not change for 3 epochs.

### Evaluation Metrics
We report the results obtained by averaging 5 runs with different seeds. We calculate BLEU scores with case-sensitive tokenization using Moses, and apply the compound splitting BLEU for WMT, the same as HAT. We test the latency of models on an Intel Xeon Silver 4114 CPU and an NVIDIA GTX 1080Ti GPU.

| Method                        | Latency (CPU) | Latency (GPU) | PPL  |
|-------------------------------|---------------|---------------|------|
| Baevski and Auli (2019)       | 12.49         | 0.53          | 18.70|
| Dai et al. (2019)             | 11.23         | 0.42          | 18.30|
| Press et al. (2020)           | 12.17         | 0.52          | **17.96**|
| RankNAS (Ours)               | **4.83**      | **0.29**      | 18.13|

Table 2: Performance of our discovered model and the state-of-the-art language models. The perplexities are evaluated on the WikiText-103 test data. Latency is measured in units of seconds. All models have a similar size, around 250M.
### 4.2 Results

#### Hardware-Aware Architecture Search.

The hardware-aware NAS aims to maximize the accuracy under specified latency constraints on different hardware platforms. We first rank architectures by their latencies and pick those that meet the constraint to achieve this goal. Then we rank the selected architectures by their losses on the validation set and choose the best one. For machine translation tasks, we use the same search space as HAT (Wang et al., 2020), which contains around $10^{15}$ possible architectures. For the language modeling task, we use the following search space: $[10, 12, 14]$ for decoder layer number, $[768, 1024]$ for embedding dimension, $[3072, 4096, 5120]$ for hidden dimension, and $[8, 12, 16]$ for the head number in attention modules. We add a simple linear projection without bias if two adjacent layers have different hidden sizes.

Table 1 shows the results of RankNAS comparing to HAT (Wang et al., 2020) and Transformer (Vaswani et al., 2017) on the machine translation tasks. Our method is effective in reducing the search cost for different tasks and hardware platforms. For instance, it requires $10.53 \times$ less cost to find a comparable architecture on the WMT task. The discovered architectures also have the lowest latencies with the same or better BLEU scores on most tasks. For example, the architecture designed for the CPU is $2.68 \times$ faster than the standard Transformer.

We present the architecture search results for language modeling on the WikiText-103 test data in Table 2. All models are evaluated with a context window of 2,560 tokens, following Baevski and Auli (2019). Our method significantly accelerates the baseline on different devices. Specifically, our method speeds up the baseline by $2.59 \times$ on the CPU and $1.83 \times$ on the GPU. Our model also obtains a perplexity of 18.13, which outperforms Transformer-XL (Dai et al., 2019) and is comparable to the state-of-the-art language model, e.g., Sandwich-Transformer (Press et al., 2020).

#### High-Accuracy Architecture Search.

Unlike hardware-aware architecture search, the high-accuracy architecture search only optimizes accuracy and does not consider latency. In the experiments, we enlarge the HAT’s search space by introducing two additional features Relative Attention Position (Shaw et al., 2018) and Layer Norm Position, as shown in Table 5 and Table 6. This expands the size of search space to $10^{23}$, 8 orders.
We analyze both the accuracy and efficiency of our search method and study the effect of different features on model performance.

5 Analysis

5.1 Architecture Ranking Accuracy

To study the accuracy of the proposed method, we evaluate it on the IWSLT translation task. We randomly sample 200 different architectures from the HAT search space (small) and the enlarged search space (large) introduced in Sec. 4.2. We train these architectures from scratch and measure their BLEU scores on the test set. Table 4 presents the Kendall and Spearman rank correlation coefficient between the predicted results and the real scores. RankNAS outperforms HAT in terms of different ranking correlations. For example, RankNAS achieves a high Kendall’s Tau of 0.883 and 0.826 on small and large spaces. This indicates that the predicted ranking is very close to the real results.

| Search Space | Method | Kendall’s $\tau$ | Spearman’s $\rho$ |
|--------------|--------|------------------|------------------|
| Small        | HAT    | 0.827            | 0.913            |
|              | Ours   | **0.883**        | **0.945**        |
| Large        | HAT    | 0.754            | 0.842            |
|              | Ours   | **0.826**        | **0.907**        |

Table 4: RankNAS vs. HAT in terms of Kendall and Spearman rank correlation coefficient. The results are collected using the settings described in Sec. 5.1.

We compare RankNAS with state-of-the-art machine translation models designed by human experts and models discovered by other NAS methods. The results are presented in Table 3. RankNAS consistently outperforms other methods in both the IWSLT and WMT tasks. It demonstrates that RankNAS can also design high-accuracy architectures. Notably, the discovered architectures achieve a +1.8 BLEU improvement on the IWSLT task and a +1.5 BLEU improvement on the WMT task than the standard Transformers baseline (Vaswani et al., 2017). We show that RankNAS surpasses the Evolved Transformer (So et al., 2019), with orders of magnitude fewer search costs. RankNAS also matches the performance of gradient-based methods, including NAO (Fan et al., 2020) and DARTSformer (Zhao et al., 2020).

5.2 Analysis of Discovered Architectures

We present the discovered architectures in Appendix A.2 and analyze important features for different hardware on the IWSLT’14 De-En task.

Figure 5 (top) plots the selected features for the CPU. It shows that the decoder FFN dimension is the most important feature for predicting latency, followed by the decoder’s arbitrary attention and the encoder FFN dimension. We also find that the decoder embedding dimension has a similar impact on latency as the number of decoder layers.

Figure 5 (bottom) illustrates the results for the GPU. Similar to the CPU, the latency on the GPU has a high correlation to the decoder attention module. The main difference is that the latency on GPU is insensitive to FFN or embedding dimensions but more sensitive to the number of decoder layers.

The results indicate that we can design “shallow and wide” models for GPUs and “deep and thin” models for CPUs to achieve the Pareto-optimal state. Similar design insights have been verified in recent works, such as Wang et al. (2019), Hu et al. (2020), Li et al. (2021), and Lin et al. (2021).
5.3 Search Efficiency

Experiments in Sec. 4 show that our method has much lower search costs than previous works. We now analyze how does our method accelerates the architecture search.

Ranking Model Training Efficiency. The overall search cost includes the training time of the ranking model and the cost of the search process. Figure 1 compares our method and HAT on the IWSLT'14 De-En task. The two methods share the same search space and sampling strategy for search. We observe that the ranking model training takes most of the time. RankNAS speeds up the ranking model training by 10.34 times compared with HAT. Pruning the search space further reduces the 75% time of the search process. Thus the overall search cost is significantly reduced. It indicates that efficient training of the ranking model is essential to accelerate the search process.

Architecture Search Efficiency. We also analyze the efficiency of our proposed methods on the IWSLT hardware-aware task. Figure 7 shows the loss curves on the validation set of the models found by our method with different sampling strategies. We observe that RankNAS is compatible with different strategies. Also, the evolutionary algorithm outperforms random search in terms of the rate of convergence and the search result.

6 Related Work

Many efforts have been made to improve the NAS efficiency for different tasks (Tan et al., 2019; Wu et al., 2019a; Cai et al., 2019; Lu et al., 2019; Chen et al., 2020). A common approach to accelerating the search process is to use a proxy, such as reduced model size, training data, or training steps. However, it is inaccurate for estimating the model’s performance and diminishes the NAS quality (Baker et al., 2018; Dudziak et al., 2020). Another popular way is to share parameters among all architectures to reduce the training time (Tan et al., 2019; Cai et al., 2019). However, it is infeasible to train all architecture candidates fairly to obtain their accurate performance.

Recent works explored performance prediction based on architectural properties, i.e., the network topology and the model size (Liu et al., 2018; Long et al., 2019; Wen et al., 2020; Ning et al., 2020). For instance, Hardware-Aware Transformer (HAT) (Wang et al., 2020) encoded architectures into feature vectors and predicted the latency with a Multilayer Perceptron (MLP) for the target hardware. BRP-NAS (Dudziak et al., 2020) proposed an end-to-end performance predictor based on a Graph Convolutional Network (GCN). Although these methods greatly improve the performance estimation efficiency, they still require many samples and train numerous neural networks to converge, thereby increasing the search cost. Instead, we are motivated by the fact that NAS is expected to distinguish different candidate architectures. Thus, NAS can be solved by learning pairwise ranking rather than obtaining the accurate performance of architectures.

7 Conclusion

We have presented RankNAS, a simple yet efficient NAS algorithm for both hardware-aware and high-accuracy architecture search. We have shown that pairwise ranking can significantly improve search efficiency. We also have proposed a search space pruning method to help the ranking model be more efficient during the search. Our approach outper-
forms prior methods in both efficiency and accuracy. RankNAS requires 80% less time in ranking model training on the hardware-aware search task and accelerates the overall search process by 11.53 times. Also, the architectures discovered by our method outperform state-of-the-art Transformer models in terms of efficiency and accuracy.

Acknowledgements

This work was supported in part by the National Science Foundation of China (Nos. 61876035 and 61732005), the National Key R&D Program of China (No.2019QY1801), and the Ministry of Science and Technology of the PRC (Nos. 2019YFF0303002 and 2020AAA0107900). The authors would like to thank the anonymous reviewers for their comments and suggestions.

References

Alexei Baevski and Michael Auli. 2019. Adaptive input representations for neural language modeling. In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net.

Bowen Baker, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. 2018. Accelerating neural architecture search using performance prediction. In 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings. OpenReview.net.

Leo Breiman. 2001. Random forests. Mach. Learn., 45(1):5–32.

Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. 2005. Learning to rank using gradient descent. In Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005, volume 119 of ACM International Conference Proceeding Series, pages 89–96. ACM.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Efficient architecture search by network transformation. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAIA-18), New Orleans, Louisiana, USA, February 2-7, 2018, pages 2787–2794. AAAI Press.

Han Cai, Ligeng Zhu, and Song Han. 2019. Proxylessnas: Direct neural architecture search on target task and hardware. In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net.

Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. 2020. Adabert: Task-adaptive BERT compression with differentiable neural architecture search. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, pages 2463–2469. ijcai.org.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 2978–2988, Florence, Italy. Association for Computational Linguistics.

Łukasz Dudziak, Thomas C. P. Chau, Mohamed S. Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas D. Lane. 2020. BRP-NAS: prediction-based NAS using gcns. In Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual.

Yang Fan, Fei Tian, Yingce Xia, Tao Qin, Xiang-Yang Li, and Tie-Yan Liu. 2020. Searching better architectures for neural machine translation. IEEE ACM Trans. Audio Speech Lang. Process., 28:1574–1585.

Aaron Fisher, Cynthia Rudin, and Francesca Dominici. 2019. All models are wrong, but many are useful: Learning a variable’s importance by studying an entire class of prediction models simultaneously. J. Mach. Learn. Res., 20:177:1–177:81.

Chi Hu, Bei Li, Yinqiao Li, Ye Lin, Yanyang Li, Chenglong Wang, Tong Xiao, and Jingbo Zhu. 2020. The NiuTrans system for WNMT 2020 efficiency task. In Proceedings of the Fourth Workshop on Neural Generation and Translation, pages 204–210, Online. Association for Computational Linguistics.

Yufan Jiang, Chi Hu, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. 2019. Improved differentiable architecture search for language modeling and named entity recognition. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 3585–3590, Hong Kong, China. Association for Computational Linguistics.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 3146–3154.
A Appendix

A.1 High-Accuracy Architecture Search Space

Other design choices are adopted from HAT’s search space (Wang et al., 2020) with slight modifications. Inspired by Shaw et al. (2018), we search for the maximum relative position (RPR Len) in the self-attention modules in each layer. As suggested by Wang et al. (2019) and Xiong et al. (2020), proper locations of layer normalization lead to better performance. Therefore, we let NAS decide whether to put the layer normalization inside (Pre-LN) or between (Post-LN) the residual blocks.

| Features          | Search Space                                      |
|-------------------|---------------------------------------------------|
| Enc Layer Num     | [6]                                               |
| Enc Emb Dim       | [512, 640, 768]                                   |
| Enc FFN Dim       | [768, 1024, 1536, 2048]                           |
| Enc Head Num      | [2, 4, 8]                                         |
| Enc RPR Len       | [8, 12, 16]                                       |
| Enc Norm Type     | [Pre-LN, Post-LN]                                |
| Dec Layer Num     | [1, 2, 3, 4, 5, 6]                                |
| Dec Emb Dim       | [512, 640, 768]                                   |
| Dec FFN Dim       | [768, 1024, 1536, 2048]                           |
| Dec Head Num      | [2, 4, 8]                                         |
| Dec RPR Len       | [8, 12, 16]                                       |
| Enc-Dec Attn      | [Pre-LN, Post-LN]                                |

Table 5: The search space for high-accuracy search on the IWSLT’14 De-En translation task.

| Features          | Search Space                                      |
|-------------------|---------------------------------------------------|
| Enc Layer Num     | [6]                                               |
| Enc Emb Dim       | [640, 768, 1024]                                  |
| Enc FFN Dim       | [2048, 3072, 4096, 5120]                          |
| Enc Head Num      | [4, 8, 16]                                        |
| Enc RPR Len       | [8, 12, 16]                                       |
| Enc Norm Type     | [Pre-LN, Post-LN]                                |
| Dec Layer Num     | [1, 2, 3, 4, 5, 6]                                |
| Dec Emb Dim       | [640, 768, 1024]                                  |
| Dec FFN Dim       | [2048, 3072, 4096, 5120]                          |
| Dec Head Num      | [4, 8, 16]                                        |
| Dec RPR Len       | [8, 12, 16]                                       |
| Enc-Dec Attn      | [Pre-LN, Post-LN]                                |

Table 6: The search space for high-accuracy search on the WMT’14 En-De translation task.

A.2 Visualization of Good Architectures

Figure 8 illustrates one of the discovered Transformer architecture. The presented architecture achieves 36.2 BLEU on the IWSLT’14 De-En translation task and has a latency of 77ms on the GTX 1080Ti GPU, outperforming the vanilla Transformer by +1.8 BLEU and 2.6 times speed.