TA-GATES: An Encoding Scheme for Neural Network Architectures

Xuefei Ning[1,2,†,*] **Zixuan Zhou**[1,*] **Junbo Zhao**[1] **Tianchen Zhao**[1] **Yiping Deng**[2]

Changcheng Tang[3] **Shuang Liang**[3] **Huazhong Yang**[1] **Yu Wang**[1,†]

Department of Electronic Engineering, Tsinghua University[1]
TCS Lab, Huawei[2]
Novauto Technology Co. Ltd.[3]

Abstract

Neural architecture search tries to shift the manual design of neural network (NN) architectures to algorithmic design. In these cases, the NN architecture itself can be viewed as data and needs to be modeled. A better modeling could help explore novel architectures automatically and open the black box of automated architecture design. To this end, this work proposes a new encoding scheme for neural architectures, the Training-Analogous Graph-based ArchiTecture Encoding Scheme (TA-GATES). TA-GATES encodes an NN architecture in a way that is analogous to its training. Extensive experiments demonstrate that the flexibility and discriminative power of TA-GATES lead to better modeling of NN architectures. We expect our methodology of explicitly modeling the NN training process to benefit broader automated deep learning systems. The code is available at `https://github.com/walkerning/aw_nas`.

1 Introduction

The past decade has witnessed tremendous advances of deep learning (DL) methods using neural networks (NNs). One of the key driving forces behind NN's widespread applications is the clever design of NN architectures that are both effective and efficient [15, 37, 11, 13]. Nevertheless, long-tail distributed tasks and platforms in the real world pose challenges to the scalability of manual architecture design workflows. One pathway to tackle these challenges is to construct Automated Deep Learning (AutoDL) systems [38, 32, 9] that can decide on factors in all aspects of NN training and inference in an automated and algorithmic way. Neural Architecture Search (NAS) [57, 8, 46] is a subfield of AutoDL that focuses on automated architecture exploration.

As NAS shifts up the architecture design level from manual design to algorithmic design, NN architectures themselves can be viewed as data and need to be modeled. For example, predictor-based NAS [27, 23, 42, 20] constructs a performance predictor consisting of an architecture encoding scheme and a prediction head, and uses its predictions for unseen architectures to guide architecture sampling. With a good architecture encoding scheme, we can explore architecture design spaces efficiently by only sampling promising architectures [23, 27, 43]. Moreover, an encoding scheme can improve parameter-sharing evaluation [55] or help open the black box of automated architecture design [33]. To this end, our work aims at designing a better encoding scheme for NN architectures.

State-of-the-art (SOTA) encoding schemes [54, 27] of NN architectures are graph-based ones that view an architecture as a directed acyclic graph (DAG) of computations. As shown in Fig. 1 and described

[*]Equal contribution.
[†]Corresponding to: foxdoraame@gmail.com, yu-wang@tsinghua.edu.cn

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

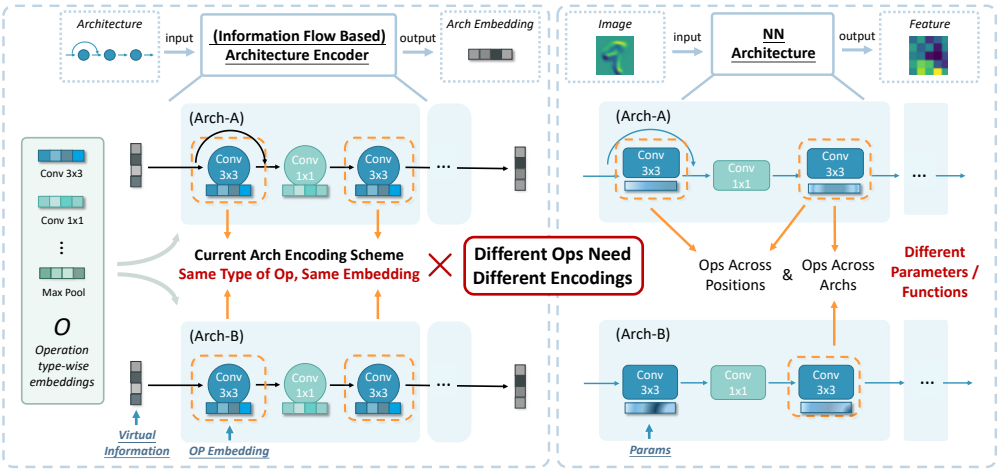


Figure 1: Motivation illustration. **Left**: State-of-the-art information flow based encoders (architecture as input, architecture encoding as output) [54, 27]. In their encoding process, all Conv3x3 share the embedding in O, no matter which architecture or which position the operation is in. **Right**: NN architecture (data as input, feature as output). Conv3x3 operations across positions (e.g., the two Conv3x3 in Arch-A) and across architectures (e.g., the first Conv3x3 in Arch-A and Arch-B) obtain different parameters through NN training represent different functions. For example, different from a plain Conv3x3, a Conv3x3 with a surrounding skip connection represents a residual function.

in Sec. A.1, these encoding schemes map each type of computation or operation (e.g., MaxPool, Conv3x3) to the same embedding, and the operation type-wise embeddings O are trainable parameters of the encoder. And their encoding process of an architecture DAG could be seen as a mimicking of how the architecture handles real data. More concretely, in the real NN forward propagation, the images are put at the input node, flow through the architecture, get processed by each operation, and output feature maps at the final output node. While in the encoding process of GATES [27], analogically, the input node embedding flows through the architecture DAG, gets multiplied by the embedding corresponding to each operation, and outputs the architecture encoding at the final output node. This encoding process mimicking the NN forward propagation is shown to outperform non-graph-based encoding schemes [23, 42, 20, 47] and vanilla GCN-based schemes [35, 10].

However, these two schemes neglect an important characteristic of NN architectures: an architecture is not a DAG with fixed computations (e.g., $\sin$, $\cos$, MaxPool), but a DAG with trainable operations (i.e., parameterized operations, e.g., Conv3x3). That is to say, operations of the same type (e.g., two Conv3x3) can represent different functions since they obtain different parameters through NN training. When encoding an architecture, the SOTA encodings [54, 27] make all operations of the same type across different locations and architectures use the same embedding, which does not take account of this characteristic of architectures. Fig. 1 illustrates that we need a scheme that can give contextualized embeddings for operations according to their architectural context.

To realize this, we base our design on the fundamental idea that an NN architecture not only depicts the computations in a forward propagation, but also implies its learning dynamics. In view of this, we propose an intuitive method, Training-Analogous Graph-based ArchiTecture Encoding Scheme (TA-GATES). During the encoding process of an architecture, TA-GATES mimics the learning dynamics of its parameterized operations. This training-analogous modeling enables TA-GATES to better encode NN architectures. We summarize the contributions of this paper as follows.

- We propose TA-GATES, a novel encoding scheme for NN architectures with analogous modeling of the NN training process. To encode an architecture DAG, TA-GATES conducts an iterative process of forward and backward passes on the DAG for several time steps, and uses the forward output at the final time step as the architecture encoding. In each time step, TA-GATES updates the embeddings of parameterized operations. This encoding process that iteratively updates the operation embeddings could be seen as a mimicking of the architecture training process that iteratively updates the operation parameters. In this way, operations get contextualized embeddings according to their architectural context.

- To further improve the discriminative power of TA-GATES (see Sec. 3.2), we propose symmetry-breaking techniques for operation embeddings at the beginning of the encoding process (before the iterative time steps). This technique could be seen as a mimicking of the random parameter initialization in the actual architecture training process.
- We apply TA-GATES for performance prediction in various architecture benchmarks. Experiments show that TA-GATES consistently surpasses baseline schemes across search spaces, for different tasks (ranking, regression, anytime prediction) and under different settings.

2 Related Work

Automated Deep Learning. In the complex pipeline to build DL models, there are many steps and components that require substantial expert knowledge. The need to design DL models for vast tasks and platforms has aroused research interests in AutoDL. NAS [8, 46] is proposed to automatically design the NN architecture. As for other training-time design choices, hyper-parameter optimization (HPO) [38] is a long-standing topic that dates back to 1990s [56]. AutoAug [3, 4] aims at designing data augmentation. And AutoLoss methods [18, 17] automate the design of loss functions.

Architecture Benchmarks. Topological architecture search spaces can be classified into the operation-on-node (OON) ones and the operation-on-edge (OOE) ones. Architectures in OON or OOE spaces have operations on their nodes or edges, respectively. Researchers have established many benchmarks for the ease of comparing NAS methods [51, 52, 7, 30, 36, 50]. Benchmarks on OON spaces include NAS-Bench-101 (NB101) [51], NAS-Bench-1Shot1 [52]. And as for OOE spaces, NAS-Bench-201 (NB201) [7] is a benchmark on a small search space of one cell architecture. NDS ENAS [30] and NAS-Bench-301 (NB301) [36] provide benchmarks on larger search spaces [29, 21] of two cell architecture (i.e., the normal and reduce cells).

Predictor-based NAS. A NAS method consists of two components: (1) The evaluation strategy gives out the performance of an architecture, e.g., by training on the training dataset to get its parameters and then testing on the validation dataset [57]. (2) The search strategy samples architectures to evaluate and explores new architectures according to the feedback of the evaluation strategy.

To improve the exploration efficiency in the large space, the predictor-based search strategy [27, 23, 42, 20] trains an approximate performance predictor of architectures using some pairs of architecture and ground-truth (GT) performance, and uses its prediction scores of unseen architectures to guide architecture sampling. An accurate predictor helps sample architectures more likely to be well-performing and improves NAS results. Lots of efforts have been devoted to developing the architecture encoding scheme, as it is essential for the predictor's generalization ability to unseen architectures.

Architecture Encoding Schemes. Existing encoding schemes of NN architectures include non-graph-based ones and graph-based ones. Non-graph-based schemes convert the computational DAG into a sequence [23, 42, 20, 47, 22, 35, 44] or image [47], and apply XGBoost, Multi-Layer Perceptron (MLP), Long Short-Term Memory (LSTM), or Convolutional Neural Network (CNN). They do not explicitly use the graph topology, and have notable weakness such as improper isomorphism handling.

As for graph-based schemes, there are attempts [35, 10] to apply plain graph convolutional networks (GCNs) [14] to encode architectures. As plain GCNs cannot be applied to OOE spaces directly, to encode architectures in OOE spaces, these methods either adopt the line-graph conversion trick [35, 45] or propose ad-hoc solutions [10] to convert architectures into OON graphs. NASBOWL [33] proposes to use the WL graph kernel with multiple iterations in the Gaussian Process surrogate. These schemes are not dedicatedly designed for NN architectures. SOTA graph-based schemes, D-VAE [54] and GATES [27], view an NN architecture as a computational DAG and follow its "information flow" to encode it. Their analogous modeling of the NN forward propagation provides better encoding.

Another related work, CATE [49], proposes a transformer-based encoder. The motivation behind CATE's transformer design is to capture "deep contextualized information" of operations. They use reachability-masked attention to aggregate operation embeddings and get contextualized operation embeddings. In this work, we propose a more intrinsical way to get contextualized operation embeddings, which can also enable more potentials such as anytime prediction.

3 Training-Analogous GATES

State-of-the-art encoding schemes [54, 27] view an NN architecture as a computational DAG on which the information flows and gets processed. We refer to them as the information flow-based schemes. As shown in Fig. 1 (left), in their encoding process, a piece of virtual information (a trainable encoder parameter E in Alg. 1) is used as the input node embedding of the architecture DAG. Then, this information flows along the DAG, and when the information arrives at an operation, the information is transformed according to the embedding of that operation as shown in Equ. A2 and Equ. A3. Finally, the information at the output node is adopted as the architecture embedding (see also Sec. A.1). Their encoding process of an architecture is analogous to its data processing.

We notice that they neglect an important characteristic of NN architectures: An NN architecture is not a DAG with fixed computations, but a DAG with trainable operations. These trainable operations are parametrized, and their parameters are obtained through a training process, which implies that two operations of the same type with different parameters represent different functions.

As discussed in Fig. 1, neglecting this characteristic can result in improper modeling of operations and architectures. Is there an intrinsical way to solve this issue? Based on the fundamental idea that an NN architecture not only depicts the computations in the forward propagation but also implies the learning dynamics, TA-GATES encodes architectures by mimicking their training process.

In the following, Sec. 3.1 first describes the encoding process of TA-GATES. Then, we elaborate on our proposed "symmetry-breaking" technique in Sec. 3.2. And Sec. 3.3 describes how the explicit modeling of the training process can empower the anytime performance prediction task.

3.1 Iterative Encoding in Analogy to Iterative Parameter Training

We give out the encoding process of TA-GATES in Alg. 1 (Line 1-10) and an illustration in Fig. 2 (upper). The learnable parameters and modules of TA-GATES and notations are also summarized in the first two sections in Alg. 1. We break down the encoding process as follows. First, we get the initial operation embedding $\text{emb}_{\text{op}}^{\text{ori}} = \text{GetEmbedding}(O, op) \in \mathbb{R}^{M \times d_o}$ for all M operations in the DAG according to their operation types (Line 1). Note that $O \in \mathbb{R}^{N_o \times d_o}$ is an operation type-wise embedding matrix, where each row corresponds to one operation type. At this point, operations of the same type have the same embeddings in $\text{emb}_{\text{op}}^{\text{ori}}$. Then, we apply SymmetryBreaking on $\text{emb}_{\text{op}}^{\text{ori}}$ to get $\text{emb}_{\text{op}}^{(0)}$ (Line 2), which will be discussed in detail in Sec. 3.2.

Then, as shown in Fig. 2 (upper left), we conduct an iterative process of forward and backward passes on the DAG α for T time steps, in which the operation embeddings emb_{op} get iteratively updated. This process can be seen as mimicking the iterative parameter updates of the architecture in actual training. In each time step t, we first compute the information flow based GCNs on the architecture DAG a (Line 4-5). Specifically, we use a global trainable parameter E as the input information of the architecture DAG (i.e., feed E into the input node of a), and call the InfoPropagation procedure with the current operation embedding $\text{emb}_{\text{op}}^{(t-1)}$. InfoPropagation is described in Sec. A.1 and implemented following [27, 54]. After the forward GCN pass, we convert the output information of the forward pass $f_{\text{info}}^{(t)}[N]$ by applying FBConvert (an MLP) to get $b_{\text{info}}^{(t)}[N]$. Then $b_{\text{info}}^{(t)}[N]$ is used as the input information of the backward GCN pass in Line 7. Note that $b_{\text{info}}^{(t)}[N]$ is fed into the output node of a, i.e., the input node of a^T. We call the two GCN passes in Line 5/7 the forward and backward passes as they propagate on the DAG a and its transposed DAG a^T, respectively.

After the forward and backward GCN pass in the t-th time step, TA-GATES computes the operation embedding updates using the propagated information in the forward and backward passes (Line 8). Specifically, GetOpEmbUpdate concatenates the operation embedding from the last time step $\text{emb}_{\text{op}}^{(t-1)}$ and the propagated information $f_{\text{info}}^{(t)}, b_{\text{info}}^{(t)}$ in the forward and backward passes as the input, feed it into an MLP, and get the operation embedding updates $\delta^{(t)}$. Then, $\delta^{(t)}$ is updated onto the operation embeddings to get $\text{emb}_{\text{op}}^{(t)}$ (Line 9). And the updated operation embeddings $\text{emb}_{\text{op}}^{(t)}$ will be used in the following time step $t+1$.

After T time steps, the output of the T-th forward InfoPropagation pass (i.e., the node embedding at the output node) $f_{\text{info}}^{(T)}[N]$ is used as the architecture encoding.

Algorithm 1 The Encoding Process of TA-GATES

Learnable Parameter or Modules of TA-GATES:

E: the input information
W^f, W^b: the parameters of forward and backward GCNs (see Sec. A.1)
$O \in \mathbb{R}^{N_o \times d_o}$: the embeddings of N_o types of operations
FBConvert: an MLP module to convert the forward pass' output to the backward pass' input
GetOpEmbUpdate: an MLP module to get updates of operation embeddings
InfoPropagation: the information flow based GCN computation (see Sec. A.1)
SymmetryBreaking: symmetry breaking using parameter-level zero-cost metrics (see Equ. 1)

Other Notations:

T: the number of time steps
N: the number of nodes in a cell architecture
M: the number of operations in a cell
N_o: the number of operation choices in the search space (the subscript o denotes operation)
d_o: the dimension of operation embeddings (the subscript o denotes operation)
$\text{emb}_{\text{op}}^{(t)}$: the operation embeddings at time step t
$f_{\text{info}}^{(t)}[n],\ b_{\text{info}}^{(t)}[n]$: the information of the n-th node in the forward or backward pass at time step t ($n = 1$ denotes the input node, $n = N$ denotes the output node)
s: a scale of operation embeddings' updates, set to 0.1 in all experiments except ablation studies

Input:

a: a DAG denoting the NN architecture, a^T denotes its transposed DAG with all edges reversed
$op \in \{0, \cdots, N_o\}^M$: the operation indexes in a

Encoding Process:

$\text{emb}_{\text{op}}^{\text{ori}} = \text{GetEmbedding}(O, op)$
$\text{emb}_{\text{op}}^{(0)} = \text{SymmetryBreaking}(\text{emb}_{\text{op}}^{\text{ori}})$
for $t = 1, \cdots, T$ **do**
  $f_{\text{info}}^{(t)}[1] = E$
  $f_{\text{info}}^{(t)}[2:N] = \text{InfoPropagation}(f_{\text{info}}^{(t)}[1]; a, \text{emb}_{\text{op}}^{(t-1)}, W^f)$
  $b_{\text{info}}^{(t)}[N] = \text{FBConvert}(f_{\text{info}}^{(t)}[N])$
  $b_{\text{info}}^{(t)}[1:N-1] = \text{InfoPropagation}(b_{\text{info}}^{(t)}[N]; a^T, \text{emb}_{\text{op}}^{(t-1)}, W^b))$
  $\delta^{(t)} = \text{GetOpEmbUpdate}([\ \text{emb}_{\text{op}}^{(t-1)} \mid f_{\text{info}}^{(t)} \mid b_{\text{info}}^{(t)}\ ])$
  $\text{emb}_{\text{op}}^{(t)} = \text{emb}_{\text{op}}^{(t-1)} + s \times \delta^{(t)}$
end for

Output: $f_{\text{info}}^{(T)}[N]$ ($\{f_{\text{info}}^{(t)}[N]\}_{t=1,\cdots,T}$ if doing anytime performance prediction)

Analogy to NN training. The analogy between the training process and the TA-GATES's T-step iterative encoding process of an NN architecture is illustrated in Fig. 2 (upper). In each time step of TA-GATES, the forward and backward GCN passes (InfoPropagation) correspond to the NN forward and backward propagation, GetOpEmbUpdate correspond to the gradient calculation w.r.t. parameters, and the updates of operation embeddings correspond to parameter updates.

Improving the modeling flexibility. The analogous modeling of NN training adds to the encoding flexibility, as it makes the embedding of each operation adaptive to the overall architecture (i.e., contextualized embedding). Thus, two operations of the same type can get distinguishable embeddings (e.g., the 0-1/0-2 Conv3x3 in Fig. 2 (upper)), which is more reasonable considering that they represent different functions in the actual NN. In this way, the architecture encoding is more discriminative.

Training of TA-GATES. We have described the encoding process (i.e., the inference) of TA-GATES. As for the training of TA-GATES's parameters (listed in the 1st section in Alg. 1), we construct a performance predictor consisting of TA-GATES and an MLP head, input the architectures, and use their GT performances as the labels for the predictor outputs $\text{MLP}(f_{\text{info}}^{(T)}[N])$ (see Sec. A.3).

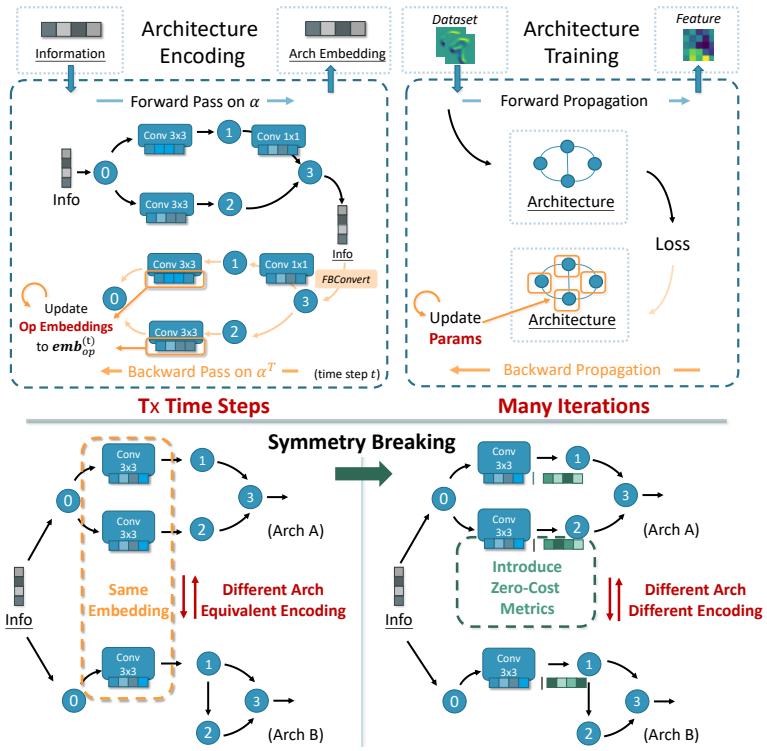


Figure 2: **Upper**: The analogy between the training (Right) of an NN architecture with 4 nodes (0/1/2/3 circles) and **the TA-GATES's iterative encoding process** (Left). **Lower**: A single forward pass in the encoding process *without* (Left) / *with* (Right) **the symmetry-breaking technique**.

3.2 Symmetry Breaking in Analogy to Random Parameter Initialization

Although the iterative encoding process in TA-GATES improves the discriminability of operations and architectures, there are still cases it fails to discriminate properly.

As shown in Fig. 2 (lower left), the two architectures are apparently different, and the upper one has a larger capacity. However, their information-based encodings [54, 27] are indistinguishable. As the three Conv3x3 in Arch-A/Arch-B have the same embeddings, and the operation between node 1 and 2 in Arch-B is a skip connection, node 1 and 2 in Arch-A and Arch-B all have the same information $f_{\text{info}}^{(t)}, b_{\text{info}}^{(t)}$. Therefore, the final architecture encoding of Arch-A and Arch-B is the same. Even if TA-GATES iteratively updates the operation embeddings in the architecture encoding process, the embeddings of these Con3x3 operations remain the same across all time steps if their initial embeddings at time step 0 are the same, since they have exactly the same paths to the input and output nodes (i.e., at fully symmetrical positions).

Actually, the reason why these two architectures are not equivalent in NN training is that the random parameter initialization breaks the symmetry of the two Conv3x3 when the training begins. Based on this analysis, we propose to apply symmetry breaking to the initial operation embeddings $\text{emb}_{\text{op}}^{(0)}$ (SymmetryBreaking in Alg. 1 Line 2) to enable TA-GATES to distinguish between symmetric operations. We propose three types of symmetry-breaking techniques:

1. **Using random noises:** The most straightforward way is to add random noises onto $\text{emb}_{\text{op}}^{(0)}$.

2. **Using zero-cost saliency metrics:** Instead of directly injecting randomness into operation embeddings, we can inject randomness into NN parameters and aggregate some metrics of parameters in each operation to refine its embedding. We propose to aggregate parameter-wise saliency metrics to break the symmetry of operation embeddings. Specifically, SymmetryBreaking in the encoding process of an architecture first randomly initializes the parameters of the architecture. Then, for each operation, we calculate and aggregate five per-parameter saliency metrics as a 5-dim vector $Z \in \mathbb{R}^{M \times 5}$, including grad_norm, snip [16], grasp [41], plain [25],

and synflow [40]. And the initial operation embedding $\text{emb}_{\text{op}}^{(0)} \in \mathbb{R}^{M \times d_o}$ is refined as

$$\text{emb}_{\text{op}}^{(0)} = \text{emb}_{\text{op}}^{\text{ori}} + \beta \times \text{MLP}^z(Z), \tag{1}$$

where the MLP^z maps from $\mathbb{R}^5$ to $\mathbb{R}^{d_o}$, and β is a fixed scale.

3. **Using zero-cost saliency metrics in every time step:** We also experiment with a variant of the 2nd technique: Instead of adding a symmetry-breaking vector onto the initial operation embedding $\text{emb}_{\text{op}}^{\text{ori}}$, we concatenate Z onto $\text{emb}_{\text{op}}^{(t)}$ in all time steps.

Though the last two techniques aggregate parameter-level metrics, they only use randomly initialized parameters and do not require actual parameter training. Thus, borrowing the terminology from zero-cost pruning and NAS [40, 1, 24, 19, 26], we refer to them as "zero-cost symmetry-breaking techniques". As illustrated in Fig. 2 (lower right), the encoding with symmetry breaking can differentiate the two operations at symmetrical positions in Arch-A, and thereby the architectures Arch-A and Arch-B.

3.3 Anytime Performance Training and Prediction

Besides improving the discriminative power of architecture encoding, the explicit modeling of NN training brings other interesting possibilities. This work explores using TA-GATES to empower anytime performance training and prediction. The meaning of the anytime prediction task is two-fold: (1) Training using performances at other epochs might help improve the prediction of final performances, as these supervisory signals bring more information without inducing additional training costs. (2) The task of predicting multiple performances across epochs has its applications, such as providing inspections into the learning dynamics, or making surrogate benchmarks [50].

The basic strategy to amend existing encoders for anytime prediction is to make it output multiple scores as the predicted performances for multiple epochs. And as it comes to TA-GATES, we have a more natural choice to use output scores of different time steps as the predicted performances, as each time step in TA-GATES corresponds to a checkpoint in the NN training process. We'll demonstrate that this natural fit of TA-GATES indeed boosts the predictive power for anytime performances.

3.4 Summary

To summarize, TA-GATES has three key steps in analogy with steps in NN training: (1) Zero-cost symmetry breaking of operation embeddings corresponds to the random initialization of parameters. (2) Forward and backward passes of GCN correspond to the forward and backward propagation of the NN architecture. (3) The updates of operation embeddings correspond to the updates of parameters.

Thanks to the analogous modeling of NN training, TA-GATES has higher flexibility and better discriminative power, and also brings interesting possibilities (see also Sec. 6). For example, TA-GATES provides a more natural and stronger solution. for the anytime performance prediction.

4 Experiments

We compare the predictive power of TA-GATES with baseline encoding schemes on four search spaces, including NB101 [51, 52], NB201 [7], NB301 [36], and ENAS [29]. Specifically, we use the performances provided by NAS benchmarks as the ground-truth (GT), and split the GT architecture-performance pairs into training and test splits. After training the predictor with (a subset of) the training split, we measure the predictor fitness on the test split.

Following previous work on architecture encoding, we adopt Kendall's Tau (KD) [34] as the main measure. We also use other measures, including Precision@K (P@K) [27][3], mean squared average error (MSE), and the Pearson coefficient of linear correlation (LC). We use the MSE regression loss to train predictors in anytime training and use the pairwise hinge ranking loss in all other experiments. Detailed experimental settings and runtime information can be found in Sec. B and Sec. C.2, and ablation studies can be found in Sec. C.3.

[3] Precision@K: The proportion of true top-K architectures in the predicted top-K architectures.

Table 1: Kendall's Tau of using different encoders on NB101, NB201, NB301 and NDS ENAS. The average result of 9 experiments are reported, and the standard deviation is in the subscript.

	Encoder	Proportions of 7290 training samples				
		1%	5%	10%	50%	100%
NB101	MLP [42]	$0.3937_{(0.0302)}$	$0.5318_{(0.0185)}$	$0.5703_{(0.0167)}$	$0.6225_{(0.0078)}$	$0.6307_{(0.0069)}$
	LSTM [42]	$0.5476_{(0.0341)}$	$0.5876_{(0.0245)}$	$0.6040_{(0.0154)}$	$0.6196_{(0.0142)}$	$0.6131_{(0.0185)}$
	GCN (global node) [35]	$0.3668_{(0.0563)}$	$0.5973_{(0.0233)}$	$0.6927_{(0.0108)}$	$0.7520_{(0.0075)}$	$0.7689_{(0.0083)}$
	NASBOWL [33]	$0.5850_{(0.0232)}$	$0.6416_{(0.0241)}$	$0.6536_{(0.0193)}$	$0.6833_{(0.0022)}$	$0.6872_{(0.0000)}$
	SemiNAS [22]	$0.5273_{(0.0589)}$	$0.6055_{(0.0294)}$	$0.5953_{(0.0279)}$	$0.6040_{(0.0284)}$	$0.6043_{(0.0179)}$
	XGBoost [44]	$0.4517_{(0.0470)}$	$0.5987_{(0.0365)}$	$0.5680_{(0.0125)}$	$0.5677_{(0.0077)}$	$0.6175_{(0.0000)}$
	GATES [27]	$0.6321_{(0.0251)}$	$0.7493_{(0.0166)}$	$0.7690_{(0.0077)}$	$0.7999_{(0.0071)}$	$0.8119_{(0.0071)}$
	TA-GATES	$\mathbf{0.6686}_{(0.0338)}$	$\mathbf{0.7744}_{(0.0211)}$	$\mathbf{0.7839}_{(0.0063)}$	$\mathbf{0.8133}_{(0.0053)}$	$\mathbf{0.8217}_{(0.0057)}$
	Encoder	Proportions of 7813 training samples				
		0.1%	0.5%	1%	5%	10%
NB201	MLP [42]	$0.0162_{(0.0859)}$	$0.0863_{(0.0556)}$	$0.1756_{(0.0332)}$	$0.3885_{(0.0237)}$	$0.5492_{(0.0092)}$
	LSTM [42]	$0.1935_{(0.1806)}$	$0.5079_{(0.0715)}$	$0.5691_{(0.0110)}$	$0.6690_{(0.0189)}$	$0.7395_{0.0061)}$
	Line-Graph GCN [35]	$0.2461_{0.1549)}$	$0.3113_{0.0626)}$	$0.4080_{0.0369)}$	$0.5461_{0.0138)}$	$0.6095_{0.0164)}$
	NASBOWL [33]	$0.4980_{(0.0408)}$	$0.6674_{(0.0077)}$	$0.5912_{(0.0874)}$	$0.7259_{((0.0098)}$	$0.7625_{(0.0083)}$
	XGBoost [44]	$0.0706_{(0.1238)}$	$0.3719_{(0.0560)}$	$0.4178_{(0.0288)}$	$0.6412_{(0.0053)}$	$0.7084_{(0.0123)}$
	GATES [27]	$0.4309_{0.1062)}$	$0.6702_{0.0254)}$	$0.7571_{0.0169)}$	$0.8583_{0.0019)}$	$0.8823_{0.0024)}$
	TA-GATES	$\mathbf{0.5382}_{(0.0478)}$	$\mathbf{0.6707}_{(0.0256)}$	$\mathbf{0.7731}_{(0.0249)}$	$\mathbf{0.8660}_{(0.0060)}$	$\mathbf{0.8890}_{(0.0049)}$
	Encoder	Proportions of 5896 training samples				
		0.5%	1%	5%	10%	50%
NB301	MLP [42]	$0.2750_{(0.0722)}$	$0.4018_{(0.0209)}$	$0.5373_{(0.0093)}$	$0.5687_{(0.0060)}$	$0.6249_{(0.0021)}$
	LSTM [23]	$0.5161_{(0.0446)}$	$0.5689_{(0.0218)}$	$0.6893_{(0.0047)}$	$0.7144_{(0.0032)}$	$0.7572_{(0.0019)}$
	GCN [10]	$0.0951_{(0.0350)}$	$0.1280_{(0.0441)}$	$0.2673_{(0.0061)}$	$0.2835_{(0.0059)}$	$0.3179_{(0.0013)}$
	XGBoost [44]	$0.2725_{(0.0395)}$	$0.3059_{(0.0285)}$	$0.3313_{(0.0120)}$	$0.3227_{(0.0217)}$	$0.3461_{(0.0034)}$
	GATES [27]	$0.5616_{(0.0251)}$	$0.6064_{(0.0275)}$	$0.6916_{(0.0112)}$	$0.7180_{(0.0067)}$	$0.7595_{(0.0027)}$
	TA-GATES	$\mathbf{0.5728}_{(0.0307)}$	$\mathbf{0.6351}_{(0.0138)}$	$\mathbf{0.7123}_{(0.0087)}$	$\mathbf{0.7331}_{(0.0071)}$	$\mathbf{0.7685}_{(0.0066)}$
	Encoder	Proportions of 500 training samples				
		5.0%	10.0%	25.0%	50.0%	100.0%
ENAS	MLP [42]	$0.1607_{(0.0518)}$	$0.2264_{(0.0470)}$	$0.3543_{(0.0139)}$	$0.3858_{(0.0090)}$	$0.4141_{(0.0036)}$
	LSTM [23]	$0.2594_{(0.0573)}$	$0.3406_{(0.0370)}$	$0.4509_{(0.0175)}$	$0.4875_{(0.0133)}$	$0.5517_{(0.0048)}$
	GCN [10]	$0.2301_{(0.0923)}$	$0.3140_{(0.0151)}$	$0.3367_{(0.0080)}$	$0.3508_{(0.0108)}$	$0.3715_{(0.0041)}$
	GATES [27]	$0.3400_{(0.0417)}$	$0.4286_{(0.0104)}$	$0.5274_{(0.0245)}$	$0.5971_{(0.0128)}$	$0.6467_{(0.0119)}$
	TA-GATES	$\mathbf{0.3458}_{(0.0383)}$	$\mathbf{0.4407}_{(0.0104)}$	$\mathbf{0.5485}_{(0.0251)}$	$\mathbf{0.6324}_{(0.0128)}$	$\mathbf{0.6683}_{(0.0076)}$

4.1 Comparison with Baseline Encoders

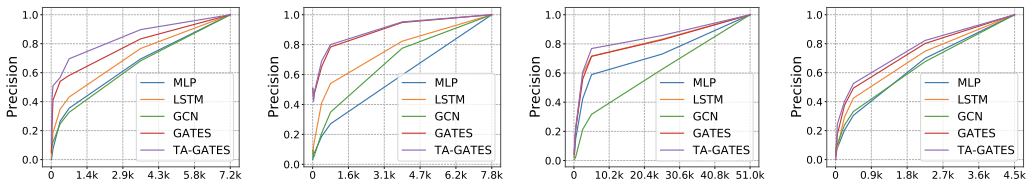
Table 1 shows the Kendal's Tau on test set using different encoders. Different columns show the results of training using different numbers of architectures. For example, a proportion of 1% on NB101 means that 1%×7290=72 architectures are used for predictor training. The comparison of P@K is shown in Fig. 3. We can see that TA-GATES achieves superior ranking quality consistently. We also conduct architecture search experiments using TA-GATES and discuss the results in Sec. C.2.

4.2 Comparison of Symmetry-Breaking Techniques

As shown in Table 2, except for two exceptions when there are only a few training architectures (39/29 training architectures on NB201/NB301), the "Add" symmetry-breaking technique brings improvements on TA-GATES without symmetry breaking ("None"), and achieves the best results. Therefore, we use the "Add" symmetry-breaking technique without explicit statements.

4.3 Anytime Performance Training and Prediction

Intuitively, the analogous encoding process of TA-GATES has a natural fit for the anytime performance prediction task. We use MSE training loss to train encoders, and show the KDs and regression measures (LC and MSE) in Table 3, Table A12, Table A13, and Table A14. We can see that TA-GATES significantly boost the anytime prediction performances.



(a) NB101 (b) NB201 (c) NB301 (d) NDS-ENAS

Figure 3: Precision@K comparison on the validation split of four benchmarks. X-axis: K; Y-axis: Precision. The training proportion is 5% on NB101, NB201, and NB301, and 50% on NDS ENAS.

Table 2: Kendall's Tau of using different symmetry-breaking techniques. "None" indicates TA-GATES without symmetry breaking. "Random", "Add", "Concat" refer to TA-GATES with the three symmetry-breaking techniques described in Sec. 3.2, respectively.

Method	NB101 (7290 training)				NB201 (7812 training)				NB301 (5896 training)			
	1%	5%	10%	50%	0.5%	1%	5%	10%	0.5%	1%	5%	10%
GATES	0.6321	0.7493	0.7690	0.7999	0.6702	0.7571	0.8583	0.8823	0.5616	0.6064	0.6916	0.7180
None	0.6510	0.7581	0.7704	0.8070	0.6838	0.7667	0.8623	0.8866	0.5735	0.6182	0.7020	0.7280
Random	0.6425	0.7612	0.7711	0.8020	0.6777	0.7688	0.8634	0.8836	**0.5735**	0.6207	0.7034	0.7257
Concat	0.6585	0.7689	0.7819	0.8086	**0.6847**	0.7708	0.8632	0.8856	0.5659	0.6230	0.7048	0.7261
Add	**0.6686**	**0.7744**	**0.7839**	**0.8133**	0.6707	**0.7731**	**0.8660**	**0.8890**	0.5728	**0.6351**	**0.7123**	**0.7331**

Table 3: Kendall's Tau of anytime training and prediction. **Upper/Lower**: Kendall's Tau with the half GT accuracy in the middle of training / the final GT accuracy. **Baselines**: "Single-" means to only use one supervisory signal to train a predictor. "Multi-" refers to basic strategy described in Sec. 3.3: The predictor outputs multiple scores and is trained with multiple supervisory signals.

KD with the half accuracy		NB101 (7290 training)				NB201 (7812 training)				NB301 (5896 training)			
Encoder	Training	1%	5%	10%	50%	0.5%	1%	5%	10%	0.5%	1%	5%	10%
Single-GATES	half	0.3636	0.3473	0.3147	0.4796	0.5882	0.6654	0.7317	0.7732	0.2028	0.2106	0.2807	0.3347
Multi-LSTM	half+final	0.0123	0.0659	0.0723	0.0518	0.3189	0.3797	0.4771	0.5285	0.1914	0.1464	0.1132	0.1187
Multi-GATES	half+final	0.2862	0.2912	0.2883	0.1413	0.5827	0.6574	0.7168	0.7745	0.1635	0.1654	0.1481	0.1263
TA-GATES	half+final	**0.3921**	**0.4615**	**0.4805**	**0.5674**	**0.6297**	**0.7110**	**0.7827**	**0.8140**	**0.2345**	**0.2322**	**0.3092**	**0.4249**
KD with the final accuracy		NB101 (7290 training)				NB201 (7812 training)				NB301 (5896 training)			
Encoder	Training	1%	5%	10%	50%	0.5%	1%	5%	10%	0.5%	1%	5%	10%
Single-GATES	final	0.3856	0.3820	0.5034	0.5903	0.4914	0.6915	0.7237	0.7806	0.1557	0.1549	0.1954	0.2432
Multi-LSTM	half+final	-0.0372	0.1028	0.2191	0.1473	0.4166	0.4795	0.5491	0.6062	**0.2046**	0.2153	0.2283	0.2318
Multi-GATES	half+final	0.3455	0.3341	0.3370	0.1818	0.6145	0.6902	0.7306	0.7989	0.1190	0.1209	0.1025	0.0868
TA-GATES	half+final	**0.5463**	**0.5850**	**0.5950**	**0.6477**	**0.6648**	**0.7363**	**0.8213**	**0.8624**	0.1963	**0.2944**	**0.2928**	**0.4178**

On one hand, TA-GATES enables the predictor to benefit from multiple supervisory signals by capturing the learning speed of architectures. Fig. A11 shows that a small architecture Arch-1-A learns faster (higher half accuracy) but ends at a low final accuracy, and a larger architecture Arch-1-B learns slower (lower half accuracy) but ends at a higher accuracy. TA-GATES can correctly predict the relative order of these two architectures both at the end or in the middle of the training process. In contrast, without explicit modeling of the learning dynamics, Multi-GATES tends to give out the same relative order for the half and final accuracies, and thus fails to make correct comparisons.

On the other hand, directly training baseline encoders with half and final accuracies simultaneously even leads to performance degradation. We observe a "trade-off" phenomenon where the prediction fitness for the half and final accuracy have opposite trends. See Sec. C.5 for detailed analyses.

Note that although our anytime training experiments use 2-step TA-GATES (T=2, t=1 for half and t=2 for final) and use the GT accuracies of two training epochs (half and final) for training, TA-GATES can be easily extended to anytime training with more time steps and more supervisory signals.

5 Conclusions

Neural architectures are data-processing DAGs with *trainable* operations. According to this nature, this work dedicatedly designs a *Training-Analogous* Graph-based ArchiTecture Encoding Scheme

(TA-GATES) for encoding NN architectures. The encoding process of TA-GATES mimics not only how the information is propagated and processed by operations during the NN inference process, but also the learning dynamics of operations during the NN training process. In this way, every operation can get "contextualized" embeddings according to its architectural context (i.e., what the overall architecture is and where the operation is). Extensive experiments in various search spaces show that TA-GATES consistently improves the performance predictions. We also show how the explicit modeling of NN training in TA-GATES empowers the anytime performance prediction task.

6 Broader Applications, Limitations, and Future Directions

Here we discuss two potential applications and one future extension direction of TA-GATES.

TA-GATES as the learning curve extrapolator for early-stop NAS. Besides predicting the anytime learning curve from an architecture description, TA-GATES also has the potential to be used as a black-box and learnable extrapolator of partial learning curves. An extrapolator can be handy for accelerating AutoML, where some NN training processes can be early stopped according to the extrapolated estimation at targeting epochs.

The existing extrapolator [6] assumes the NN learning curve can be described by some parametric function families. And during the training process of an architecture, they fit the function family parameters using the validation performances of the early learning curve, and then extrapolate to get the estimations at targeting epochs. Different from existing extrapolators, TA-GATES can take the architecture description as input, i.e., architecture-dependent. That is to say, during the extrapolation, instead of solely relying on the early learning curve points, a learnable architecture-dependent extrapolator (with per-architecture few-step adaption) can utilize the learning curve knowledge of other architectures, and thereby has the potential to make better extrapolations in a data-driven way.

TA-GATES as the joint encoder for other DL factors in AutoML. As TA-GATES explicitly mimics the NN training process, it is easy to extend TA-GATES to model other training-time factors besides the architecture in an elegant way. Thus, we expect TA-GATES to find its wide application in joint AutoDL systems, not restricting to NAS.

Here list some possibilities of integrating other factors into TA-GATES: (1) Training-time auxiliary towers [39] or architectural reparametrization [5] can be seen as introducing additional nodes and edges in the GCN passes when $t < T$, while still using the inference-time architecture when $t = T$. (2) Different data augmentation can be modeled as different conversions of E. (3) Different loss functions can be modeled as different FBConvert. (4) The influence of training hyper-parameters can be modeled by incorporating them into appropriate places in the encoding process. For example, the learning rate schedule is analogous to a schedule of s. In a word, the methodology of TA-GATES (i.e., analogous black-box modeling of the NN training process) can empower joint AutoDL systems by combining information regarding both the inference-time and training-time factors in an intrinsic way. TA-GATES could be used in joint AutoDL systems to help explore the inference-time architecture (NAS), the training-time architecture, the data augmentation (AutoAug), the loss function parameters (AutoLoss), hyper-parameters (HPO), and other factors in the complex DL pipeline.

Enabling TA-GATES to conduct cross-space comparisons. Although TA-GATES can be applied for different search spaces, it's still limited to in-space comparisons. This is because TA-GATES (1) takes the cell DAG instead of the overall architecture DAG as the input and (2) encodes only the variable choices defined by the space instead of all operation properties. Applying the training-analogous methodology of TA-GATES to develop a universal cross-space predictor [12] is an interesting future work, which can bridge the gap between zero-cost predictors (can conduct cross-space comparisons, non-satisfying predictions) and data-driven but space-specific predictors (cannot conduct cross-space comparisons, have stronger predictive power).

Acknowledgements

This work was supported by National Natural Science Foundation of China (No. U19B2019, 62171313, 61832007), Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua EE Xilinx AI Research Fund, and Beijing Innovation Center for Future Chips.

References

[1] Mohamed S. Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D. Lane. Zero-Cost Proxies for Lightweight NAS. In *International Conference on Learning Representations (ICLR)*, 2021.

[2] Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In *International Conference on Learning Representations (ICLR)*, 2020.

[3] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. In *IEEE International Conference on Computer Vision (ICCV)*, pages 113–123, 2019.

[4] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 702–703, 2020.

[5] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 13733–13742, 2021.

[6] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[7] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020.

[8] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *Journal of Machine Learning Research (JMLR)*, 20(55):1–21, 2019.

[9] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1487–1495, 2017.

[10] Yong Guo, Yin Zheng, Mingkui Tan, Qi Chen, Jian Chen, Peilin Zhao, and Junzhou Huang. Nat: Neural architecture transformer for accurate and compact architectures. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 735–747, 2019.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[12] Daniel Hesslow and Iacopo Poli. Contrastive embeddings for neural architectures. *arXiv preprint arXiv:2102.04208*, 2021.

[13] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[14] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Annual Conference on Neural Information Processing Systems (NIPS)*, volume 25, 2012.

[16] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations (ICLR)*, 2019.

[17] Chuming Li, Xin Yuan, Chen Lin, Minghao Guo, Wei Wu, Junjie Yan, and Wanli Ouyang. Am-lfs: Automl for loss function search. In *IEEE International Conference on Computer Vision (ICCV)*, pages 8410–8419, 2019.

[18] Hao Li, Chenxin Tao, Xizhou Zhu, Xiaogang Wang, Gao Huang, and Jifeng Dai. Auto seg-loss: Searching metric surrogates for semantic segmentation. In *International Conference on Learning Representations (ICLR)*, 2020.

[19] Ming Lin, Pichao Wang, Zhenhong Sun, Hesen Chen, Xiuyu Sun, Qi Qian, Hao Li, and Rong Jin. Zen-nas: A zero-shot nas for high-performance deep image recognition. In *IEEE International Conference on Computer Vision (ICCV)*, pages 347–356, 2021.

[20] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[21] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[22] Renqian Luo, Xu Tan, Rui Wang, Tao Qin, Enhong Chen, and Tie-Yan Liu. Semi-supervised neural architecture search. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 33:10547–10557, 2020.

[23] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 7816–7827. Curran Associates, Inc., 2018.

[24] Joseph Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. Neural architecture search without training. In *International Conference on Machine Learning (ICML)*, 2021.

[25] Michael C Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Annual Conference on Neural Information Processing Systems (NIPS)*, pages 107–115, 1989.

[26] Xuefei Ning, Changcheng Tang, Wenshuo Li, Zixuan Zhou, Shuang Liang, Huazhong Yang, and Yu Wang. Evaluating efficient performance estimators of neural architectures. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, volume 34, pages 12265–12277, 2021.

[27] Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based nas. In *European Conference on Computer Vision (ECCV)*, pages 189–204, 2020.

[28] Shuaicheng Niu, Jiaxiang Wu, Yifan Zhang, Yong Guo, Peilin Zhao, Junzhou Huang, and Mingkui Tan. Disturbance-immune weight sharing for neural architecture search. *Neural Networks*, 144:553–564, 2021.

[29] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning (ICML)*, pages 4095–4104. PMLR, 2018.

[30] Ilija Radosavovic, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollár. On network design spaces for visual recognition. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1882–1890, 2019.

[31] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[32] Esteban Real, Chen Liang, David So, and Quoc Le. Automl-zero: Evolving machine learning algorithms from scratch. In *International Conference on Machine Learning (ICML)*, pages 8007–8019. PMLR, 2020.

[33] Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael Osborne. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. In *International Conference on Learning Representations (ICLR)*, 2021.

[34] Pranab Kumar Sen. Estimates of the regression coefficient based on kendall's tau. *Journal of the American Statistical Association*, 63(324):1379–1389, 1968.

[35] Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James Kwok, and Tong Zhang. Bridging the gap between sample-based and one-shot neural architecture search with bonas. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 33, 2020.

[36] Julien Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. Nas-bench-301 and the case for surrogate benchmarks for neural architecture search. *arXiv preprint arXiv:2008.09777*, 2020.

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[38] Jasper Snoek, Hugo Larohelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Annual Conference on Neural Information Processing Systems (NIPS)*, 25, 2012.

[39] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[40] Hidenori Tanaka, Daniel Kunin, Daniel L Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 33, 2020.

[41] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations (ICLR)*, 2020.

[42] Linnan Wang, Yiyang Zhao, Yuu Jinnai, and Rodrigo Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1805.07440*, 2018.

[43] Colin White, Willie Neiswanger, Sam Nolen, and Yash Savani. A study on encodings for neural architecture search. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 33:20309–20319, 2020.

[44] Colin White, Arber Zela, Robin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 34:28454–28469, 2021.

[45] Wikipedia contributors. Line graph — Wikipedia, the free encyclopedia, 2004. [Online; accessed 22-July-2004].

[46] Lingxi Xie, Xin Chen, Kaifeng Bi, Longhui Wei, Yuhui Xu, Lanfei Wang, Zhengsu Chen, An Xiao, Jianlong Chang, Xiaopeng Zhang, and Qi Tian. Weight-sharing neural architecture search: A battle to shrink the optimization gap. *ACM Computing Surveys*, 54(9), oct 2021.

[47] Yixing Xu, Yunhe Wang, Kai Han, Yehui Tang, Shangling Jui, Chunjing Xu, and Chang Xu. Renas: Relativistic evaluation of neural architecture search. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4411–4420, 2021.

[48] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

[49] Shen Yan, Kaiqiang Song, Fei Liu, and Mi Zhang. Cate: Computation-aware neural architecture encoding with transformers. In *International Conference on Machine Learning (ICML)*, pages 11670–11681. PMLR, 2021.

[50] Shen Yan, Colin White, Yash Savani, and Frank Hutter. Nas-bench-x11 and the power of learning curves. *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 34, 2021.

[51] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning (ICML)*, pages 7105–7114. PMLR, 2019.

[52] Arber Zela, Julien Siems, and Frank Hutter. Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

[53] Chris Zhang, Mengye Ren, and Raquel Urtasun. Graph hypernetworks for neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

[54] Muhan Zhang, Shali Jiang, Zhicheng Cui, Roman Garnett, and Yixin Chen. D-vae: a variational autoencoder for directed acyclic graphs. In *Annual Conference on Neural Information Processing Systems (NeurIPS)*, pages 1588–1600, 2019.

[55] Zixuan Zhou, Xuefei Ning, Yi Cai, Jiashu Han, Yiping Deng, Yuhan Dong, Huazhong Yang, and Yu Wang. Close: Curriculum learning on the sharing extent towards better one-shot nas. In *European Conference on Computer Vision (ECCV)*, 2022.

[56] Marc-André Zöller and Marco F Huber. Benchmark and survey of automated machine learning frameworks. *Journal of Artificial Intelligence Research*, 70:409–472, 2021.

[57] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

[58] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Implementation Details and Discussions on TA-GATES

A.1 Implementation of the Information Flow Based GCN

The state-of-the-art information flow based GCN, GATES [27], puts a virtual information of dimension d_i at the input node of a cell architecture, and then calculates the information of other nodes following the topological order. And the computation when encountering an unary operation of type $op \in \{0, \cdots, N_o\}$ goes as follows. A soft attention mask $m \in (0,1)^{d_i}$ is firstly computed as

$$m = \sigma(\text{GetEmbedding}(O, op)\ W_o), \tag{A2}$$

where σ is the sigmoid function. And $O \in \mathbb{R}^{N_o \times d_o}$ is the embeddings of N_o types of operations, where d_o denotes the dimension of each operation embedding. GetOpEmbedding(O, o) get out the embedding for this type of operation and has a dimension of d_o, and $W_o \in \mathbb{R}^{d_o \times d_i}$ is a linear transformation matrix.

Then, the output information of this unary operation is calculated as

$$x_{\text{out}} = m \odot x_{\text{in}} W_x, \tag{A3}$$

where $\odot$ denotes the elementwise multiplication. And a summation is used to aggregate virtual information on multiple incoming edges of a node.[4] Finally, the information at the output node of the cell architecture is used as the architecture embedding. We also give a graphical illustration of the GATES encoding process in Fig. A4.

Note that we cannot directly take the results from the GATES [27] paper, since we conduct experiments on more benchmark search spaces, and we run each of our training experiments with 9 runs (3 different training seeds, and 3 different training sets). Therefore, we use their published codes and rerun all experiments for the baseline encoders.

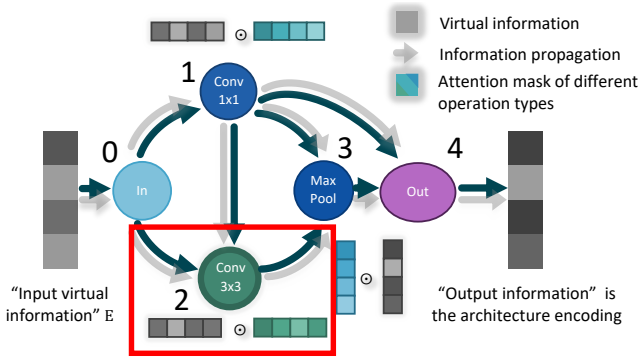


Figure A4: The encoding process of GATES [27] mimics the NN forward process. In the NN forward process, an image is taken as the input data, and each operation processes the data. In the encoding process of GATES, a piece of "virtual information" is taken as the input node embedding, and each operation is a transformation of the propagated information. For example, in the red box, the computation of the feature map F_2 at node 2 in the NN forward process is $F_2 = \text{Conv3x3}(F_0 + F_1)$. Analogically, the transformation of the "virtual information" in the encoding process at node 2 is $N_2 = m_2 \odot (N_0 + N_1)$, where the attention mask $m_2 = \sigma(\text{GetEmbedding}(O, \text{Conv3x3})W_o)$ is the attention mask corresponding to the operation type Conv3x3.

Special handling of skip connections. We add special handling of all skip connection operations into GATES and TA-GATES. Specifically, their operation embeddings are not treated as learnable parameters during the training process of TA-GATES, and their soft attention masks m are set to all 1s during the encoding process.

This handling helps identify isomorphic architectures caused by skip connections. To be more specific, let us consider the case if a node has one and only one incoming edge, on which the operation is a skip

[4] Please refer to [54, 27] for detailed discussions on how they model unary operations and aggregations.

connection. Then, this node is equivalent to its only predecessor node. This type of equivalence results in isomorphic architectures having different original DAG representations. Our special handling of the skip connection ensures the information on these two nodes are equivalent, and thus properly map isomorphic architectures caused by skip connections to the same representation. While without our special handling of the skip connections, the baseline [27] cannot guarantee to map this type of isomorphic architecture pairs to the same representation.

We empirically find that this special handling of skip connections indeed brings slight improvements on GATES [27]. Note that all our reported results of the GATES baseline have already incorporated this improvement by adding this special handling into their published codes.

In some search spaces (such as DARTS [21], NB301 [36], and ENAS [29]), the final output node in a cell architecture conducts a concatenation of some intermediate nodes. DARTS and NB301 concatenate all four intermediate nodes, while ENAS concatenates the loose-end nodes that are not used as the input for other nodes. When preparing the adjacent matrices of these cell architectures, we put "skip connection" operations on the edges between the final output node and its inputs. These skip connections are handled in the same way as the skip connections between other nodes.

A.2 Implementation of TA-GATES

In this section, we first discuss on several design choices of TA-GATES, i.e., (1) Modeling each operation as a multiplicative transform of the information propagating on graph. (2) Sharing one set of embeddings for operation types, but using different mapping W_o in the forward and backward pass. (3) The input to the GetOpEmbUpdate module. Then, we discuss how TA-GATES handles search spaces with different properties (e.g., operation on node or edge, multi-cell, multi-input-node). Note that these designs are general and correspond to search space properties, instead of being per-search-space designs.

Discussion on using GATES in the backward information propagation. In the design of TA-GATES, the backward information propagation on DAG should be analogous to the NN backpropagation. According to the multiplicative chain rule of derivatives in backpropagation, we consider that each operation plays a multiplicative role in NN backpropagation. Thus, we regard using the multiplicative GATES in the backward information propagation to be a reasonable choice, and use it in both the forward and backward information propagation on DAG.

Note that although the forward and backward information passes use the same set of operation embeddings $\text{emb}_{\text{op}}^{(t)}$ in the time step t, an operation has different transformations of information in the forward and backward passes. This is because the matrix W_o in Equ. A2 that maps the operation embedding to the attention mask m is different (W_o^f for the forward pass, W_o^b for the backward pass).

Discussion on GetOpEmbUpdate. In the NN training process, the parameter updates of each operation are influenced by other operations and the architecture topology. Thus in the encoding process, we need to input the forward and backward information into GetOpEmbUpdate to model the influence. This is why we concatenate $f_{\text{info}}^{(t)}, b_{\text{info}}^{(t)}$ together with $\text{emb}_{\text{op}}^{(t-1)}$ as the input for GetOpEmbUpdate to get $\delta^{(t)}$ in Alg. 1 Line 8.

Handling different types of search spaces. Generally speaking, there are two types of topological search spaces, operation-on-node (OON) ones [51, 52], and operation-on-edge (OOE) ones [21, 36, 29]. For completeness, we illustrate an OON and an OOE architecture in Fig. A5.[5]

TA-GATES conducts the information propagation passes with the InfoPropogation procedure, which adopts GATES [27] as the propagating method. As GATES is general to OON and OOE search space, we only discuss how to make other procedures general to the OON and OOE search spaces.

The procedures GetEmbedding, SymmetryBreaking, FBConvert, and the operation updates (Line 9) illustrated in Alg. 1 do not need any modifications to handle these two search spaces. As for the procedure GetOpEmbUpdate, for simplicity of expression, Line 8 in Alg. 1 assumes the search space is OON. Concretely, the operation embedding $\text{emb}_{\text{op}}^{(t-1)}$, its corresponding node information in the forward and backward passes are concatenated to feed into GetOpEmbUpdate. And on OOE search

[5] We follow [27] to plot this illustration.

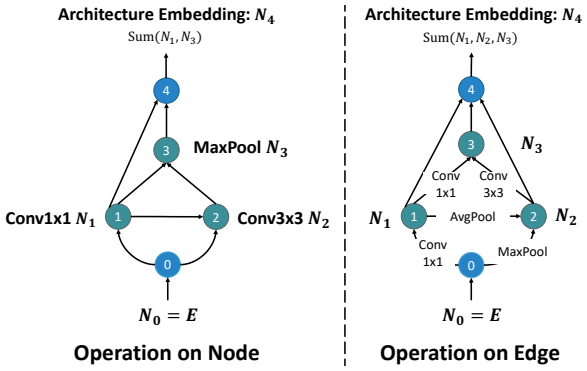


Figure A5: An illustration of example OON (left) and OOE architectures (right).

spaces, denoting the successor node of the edge e on DAG a as Successor(e, a), the embedding update for an operation on edge e is:

$$\delta^{(t)} = \text{GetOpEmbUpdate}([\,\text{emb}'_{\text{op}} \mid f_{\text{info}}^{(t)}[\text{Successor}(e, a)] \mid b_{\text{info}}^{(t)}[\text{Successor}(e, a^T)\,]) \tag{A4}$$

where emb'_{op} is the embedding of this operation in the last time step. In a word, we concatenate the information of the successor node in forward and backward passes with the operation embedding and feed the result into GetOpEmbUpdate to get the embedding update.

Handling architectures described by multiple cells. An architecture in DARTS, NB301, and ENAS search spaces is described by two cell architectures (the normal cell, the reduce cell). Alg. 1 describes the encoding process for one cell architecture. And to encode an architecture described by multiple cells, in every time step, the FBConvert procedure concatenates the output information ($f_{\text{info}}[N]$) of all cell architectures belonging to this architecture, and outputs two separate $b_{\text{info}}^{(t)}[N]$ as the input information for the backward GCN pass of the two cell architectures. The final architecture embedding is obtained by concatenating $f_{\text{info}}^{(T)}[N]$ of consisting cell architectures. All other TA-GATES parameters are shared when encoding different cell architectures.

In a word, FBConnect fuses $f\_\text{info}^{(t)}[N]$ of multiple cell architectures, and then slice it to feed into the backward InfoPropagation. In this way, the backward information and thereby the operation embedding updates in each cell architecture are related to all cell architectures, which is a more reasonable choice than independently encoding each cell architecture.

Handling multiple input nodes. For the ease of understanding, Alg. 1 only shows the case when the number of input nodes of a cell architecture equals 1. When there are more input nodes (as in DARTS, NB301, and ENAS), one just needs to increase the dimension of the learnable E to match the input node number, and initialize the information of the input nodes using E accordingly.

A.3 Training of TA-GATES

TA-GATES uses the information at the DAG output node in the last time step $f_{\text{info}}^{(T)}[N]$ as the architecture embedding. In the performance prediction task, this architecture embedding is fed into an MLP to predict a score s of this architecture. Denote $\text{E}(a)$ as the information embedding $f_{\text{info}}^{(T)}[N]$ of the architecture a, we construct the performance predictor F with a following MLP:

$$s = \text{F}(a) = \text{MLP}(\text{E}(a)) \tag{A5}$$

Following [27], we adopt a hinge pair-wise ranking loss to train the predictor.

$$L(\{a_i, y_i\}_N) = \sum_{i=1}^{N} \sum_{j=i+1}^{N} \max(0, m - (\text{F}(a_i) - \text{F}(a_j))) \tag{A6}$$

where a_i denotes one architecture, y_i denotes the corresponding ground-truth performance. m denotes the compare margin.

In the anytime performance prediction task, we use the architecture embeddings in other time step $t \neq T$ to predict other performances besides the performance at the final training epoch of the architecture. Denote $\mathrm{E}^{(t)}(a)$ as the information embedding $f_{\text{info}}^{(t)}[N]$ of the architecture a. the predicted anytime scores $\{s^{(t)}\}_{t=1,\cdots,T}$ are computed as

$$s^{(t)} = \mathrm{F}^{(t)}(a) = \mathrm{MLP}^{(t)}(\mathrm{E}^{(t)}(a)) \tag{A7}$$

In anytime training, we minimize the Mean Squared Error (MSE) between the predicted and GT performances.

$$L(\{a_i, y_i^{(0)}, y_i^{(1)}, ..., y_i^{(T)}\}_N) = \sum_{i=1}^{N}\sum_{t=1}^{T}(\mathrm{F}^{(t)}(a_i) - y_i^{(t)})^2 \tag{A8}$$

where $y_i^{(t)}$ denotes the GT performance of the architecture a_i corresponding to time step t. For example, our experiments set T to be 2, then, t=1 and t=2 correspond to the half and final GT accuracies of the architecture during its training process, respectively.

B Experimental Settings

B.1 Benchmark Split and Experiment Seeds

For the NB101 search space, we use the 14580 architecture-performance pairs provided by the 3rd subset in NAS-Bench-1shot1 [52]. For each benchmark, we split the architecture-performance pairs into the training split and the test split. The training and validation split sizes are 7290 and 7290 on NB101, 7812 and 7812 on NB201, 5896 and 51072 on NB301, and 500 and 4500 on NDS ENAS. Note that for the surrogate benchmark NB301, we only use the anchor architecture-performance pairs that are obtained through actual training and testing instead of surrogate predictions.

Denoting the size of the overall training split as $|D_t|$, given a training ratio r, 9 predictor training experiments are conducted for each encoder configuration (3 training seeds, and 3 different training set). The 3 training seeds are 21, 2021, 202121. As for the 3 different training sets, we first random shuffle the overall training split with seed 2021 and 202121, then for each training ratio r, we pick out the first $r|D_t|$ architecture-performance pairs from the original and the two shuffled training splits to train the predictor. In a word, we train the predictor on three training sets with three different training seeds, and average the measures (i.e., KD, LC, and MSE) of the last five epochs. The average result of these 9 experiments are reported.

B.2 Training of Encoders

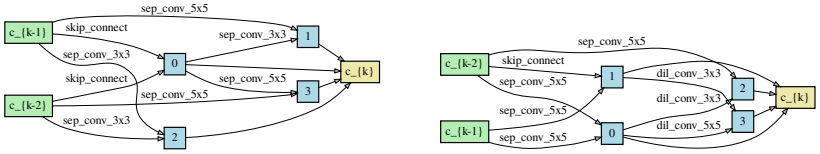
We run all predictor training and inference experiments using NVIDIA RTX 3090 GPU and AMD EPYC 7H12. For training with the ranking loss, we adopt a hinge pair-wise ranking loss with margin m=0.1, following previous studies [27, 36]. Each predictor is trained using an ADAM optimizer with a learning rate of 1e-3 and a batch size of 512 for 200 epochs.

When training with the regression loss for the anytime performance prediction task, we adopt the MSE regression loss. Each predictor is trained using an ADAM optimizer. The learning rate is set to 1e-3 on NB201 and NDS ENAS, and 1e-4 on NB101 and NB301. The number of training epochs is set to 500 on NB201 and 200 on other benchmarks. The batch size is set to 512 on all benchmarks.

We note that all training configurations are kept the same for TA-GATES and other baseline encoders.

B.3 Construction of Encoders

On NB101 and NB201, we stack five 128-dim GCN and GATES layers to construct the GCN and GATES encoders, respecitvely. Both the embedding size of the input information E and the operation embedding size are set to 48. For the GCN encoder, we follow the previous study [35] to use the feature of the global node as the architecture embedding, and adopt the line-graph conversion trick [45] to encode architectures on NB201. For TA-GATES, we construct five-layer GATES of dimension 128 for both the forward and backward propagation passes. The information and operation embedding sizes are both set to 48. The FBConvert is constructed by two 128-dim fully-connected layers. Finally, the 128-dim feature of the output node is used as the architecture embedding.



(a) Normal Cell (b) Reduction Cell

Figure A6: The discovered cell architectures by TA-GATES on NB301.

As for the MLP and LSTM baselines on NB101 and NB201, we construct the MLP encoder by 4 fully-connected layers with 512, 2048, 2048, and 512 nodes, and use the 512-dim output as the architecture's embedding. For the LSTM encoder, the embedding and hidden sizes are both set to 100, and the final hidden state is used as the architecture's embedding. We adopt the serialized representation of the architecture as the input of the MLP and LSTM encoder on NB101, following the previous study [42]. While on NB201, we use the 6 elements in the lower triangular matrix (excluding the diagonal ones) as the input.

On NB301 and ENAS, we concatenate the node and operation lists as the input of MLP and LSTM encoders following [23]. Specifically, the MLP encoder is constructed by three 128-dim fully-connected layers, and the output dimension is set to 32. For the LSTM encoder, we set the operation embedding size to 48 and the hidden size to 128. The final hidden state is used as the embedding. For GCN [10] and GATES [27], we construct the encoders by stacking 64-dim GCN and GATES layers, respectively. All the embedding sizes are set to 32. For TA-GATES, we stack 64-dim GATES layers for both the forward and backward GCN propagation. The number of GCN and GATES layer in GCN, GATES, and TA-GATES is set to 5 on NB301 and 6 on ENAS. The number of layers is set according to the maximum path length in cell architectures (5 on NB301, and 6 on ENAS). The information and operation embedding sizes are set to 32, and the FBConvert is constructed by two 64-dim fully-connected layers.

For the other baselines, we use their original codes to construct the encoders and still use the same training settings as TA-GATES. Specifically, we use NASBOWL code provided by [33] on NB101 and NB201, and use XGBoost code provided by [44] on NB101, NB201 and NB301. As for SemiNAS [22], SemiNAS actually uses an LSTM encoder. As our work focuses on the encoder design and TA-GATES should be orthogonal with the training method, we adopt their encoder construction setting using our LSTM implementation and train with our supervised training setting.

We note that we do not conduct any hyper-parameter search for TA-GATES. Only the number of TA-GATES layers is set to match the longest path in the search space. As for other configurations (e.g., number of hidden units, embedding sizes), we just set them as the same as the baseline GATES.

C Additional Experiments and Results

C.1 More Comparison Results with Baseline Encoders

Table A4 shows the Kendall's Tau with sufficient training data (i.e., 50% and 100% on NB201 and NB301) using different encoders. We can see that TA-GATES also achieves higher ranking quality under this situation.

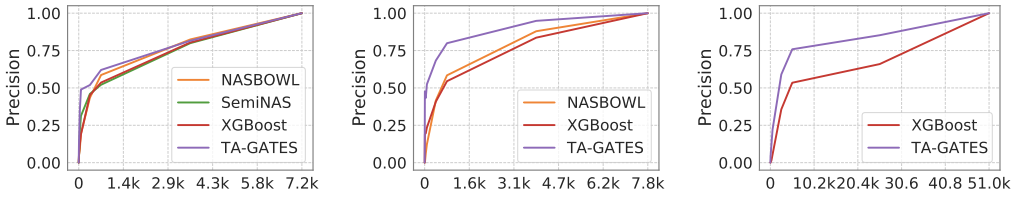
Fig. A7 shows the comparison results of P@K with more baselines on NB101, NB201, and NB301. We can see that TA-GATES still outperforms these baseline encoders. Besides, Fig. A8 shows the comparison results of P@K with sufficient training data on NB101 and NB301. The results reveal that TA-GATES can achieve better ranking quality consistently.

C.2 Architectures Search Using TA-GATES

Architecture Search on NAS Benchmarks We conduct predictor-based architecture search on three NAS benchmarks (i.e., NB101, NB201, and NB301) with different architecture encoders. After predictor training, we randomly sample 200 architectures from the search space and select

Table A4: Kendall's Tau of using different encoders on NB201 and NB301 with sufficient training data. The average result of 9 experiments are reported, and the standard deviation is in the subscript.

Method	NB201 (7812 training)		NB301 (5896 training)	
	50%	100%	50%	100%
MLP [42]	$0.8205_{(0.0050)}$	$0.8733_{(0.0011)}$	$0.6249_{(0.0021)}$	$0.6501_{(0.0014)}$
LSTM [42]	$0.8757_{(0.0018)}$	$0.9008_{(0.0013)}$	$0.7572_{0.0019)}$	$0.7672_{(0.0009)}$
GCN [35]	$0.7733_{0.0000)}$	$0.8257_{0.0000)}$	$0.3179_{0.0013)}$	$0.3256_{0.0016)}$
XGBoost [44]	$0.7977_{(0.0048)}$	$0.8312_{(0.0000)}$	$0.3461_{(0.0034)}$	$0.3766_{(0.0000)}$
GATES [27]	$0.9155_{0.0090)}$	$\mathbf{0.9259}_{0.0013)}$	$0.7595_{0.0027)}$	$0.7670_{0.0053)}$
TA-GATES	$\mathbf{0.9181}_{(0.0041)}$	$0.9228_{(0.0041)}$	$\mathbf{0.7685}_{(0.0066)}$	$\mathbf{0.7766}_{(0.0033)}$



(a) NB101 (b) NB201 (c) NB301

Figure A7: Precision@K comparison on the validation split of NB101, NB201 and NB301. X-axis: K; Y-axis: Precision. The training proportions is 1% on the three benchmarks.

the architecture with the highest predicted score. As shown in Table A5, our proposed method TA-GATES helps to discover the architectures with higher accuracy on all three benchmarks.

Table A5: Accuracy of the discovered architectures on NB101, NB201, and NB301 with different encoders.

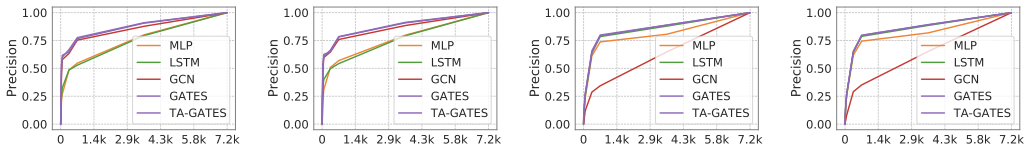
Encoder	NB101 (39 training)	NB101 (79 training)	NB101 (396 training)	NB101 (7920 training)	NB201 (78 training)	NB301 (5 training)	NB301 (58 training)	NB301 (2948 training)	NB301 (5896 training)
MLP	$0.9280_{(0.0142)}$	$0.9330_{(0.0104)}$	$0.9402_{(0.0027)}$	$0.9402_{(0.0027)}$	$0.9440_{(0.0022)}$	$0.9374_{(0.0071)}$	$0.9442_{(0.0021)}$	$0.9448_{(0.0017)}$	$0.9447_{(0.0017)}$
LSTM	$0.9359_{(0.0067)}$	$0.9388_{(0.0052)}$	$0.9416_{(0.0024)}$	$0.9434_{(0.0027)}$	$0.9334_{(0.0045)}$	$0.9380_{(0.0034)}$	$0.9440_{(0.0022)}$	$0.9444_{(0.0019)}$	$0.9442_{(0.0015)}$
GCN	$0.8750_{((0.1147)}$	$0.9026_{(0.1119)}$	$0.9417_{(0.0025)}$	$0.9434_{(0.0025)}$	$0.9244_{(0.0151)}$	$0.9254_{(0.0127)}$	$0.9214_{(0.0185)}$	$0.9384_{(0.0053)}$	$0.9378_{(0.0056)}$
GATES	$0.9407_{(0.0027)}$	$0.9420_{(0.0025)}$	$0.9417_{(0.0014)}$	$0.9428_{(0.0017)}$	$0.9349_{(0.0036)}$	$0.9398_{(0.0045)}$	$0.9454_{(0.0538)}$	$0.9450_{(0.0014)}$	$0.9448_{(0.0017)}$
TA-GATES	$\mathbf{0.9424}_{(0.0024)}$	$\mathbf{0.9426}_{(0.0022)}$	$\mathbf{0.9429}_{(0.0020)}$	$\mathbf{0.9443}_{(0.0024)}$	$\mathbf{0.9393}_{(0.0039)}$	$\mathbf{0.9417}_{(0.0045)}$	$\mathbf{0.9466}_{(0.0011)}$	$\mathbf{0.9450}_{(0.0016)}$	$\mathbf{0.9452}_{(0.0018)}$

Transferring the Picked Architecture to ImageNet. We train TA-GATES on NB301 with 10% (589) architectures of the training split, and use it to predict the scores of 10k randomly sampled architectures. We obtain the top 200 architectures based on the predicted scores, and show the architecture among them with the highest NB301 accuracy in Fig. A6.

We stack this cell architecture 14 times to construct the overall network and train it on ImageNet. The initial channel number is set to 48. We train the network for 300 epochs with batch size 256. An SGD optimizer with momentum 0.9 and weight decay 3e-5 is adopted. The learning rate is decayed from 0.1 to 0 following a cosine schedule. In the training process, the gradient norm is clipped to be less than 5, and the dropout rate is set to 0. Following previous work [58, 21], we adopt cutout augmentation with length 16 and the auxiliary tower with a weight of 0.4.

Table A6 compares the test errors of different architectures on ImageNet. As can be seen, the architecture found by TA-GATES can achieve a competitive top-1 error of 24.1% on the test set.

Discussion on the Training and Inference Time of TA-GATES. TA-GATES is more complex than GATES. Consequently, the training and inference of TA-GATES are slightly slower than GATES: On NB101, 200-epoch training of TA-GATES and GATES takes 121s and 91s (1% training architectures). Batch testing 7290 architectures with batch size 128 is very fast and only takes about 1s with both encoders. However, these costs are nearly negligible in predictor-based NAS compared to the NN training costs to get the GT accuracies of architectures (>several GPU hours per architecture),



(a) NB101 (50%) (b) NB101 (100%) (c) NB301 (50%) (d) NB301 (100%)

Figure A8: Precision@K comparison on the validation split of NB101 and NB301 with sufficient training data. X-axis: K; Y-axis: Precision. The training proportions are 50% and 100%.

Table A6: Comparison of NAS-discovered architectures on ImageNet.

Method	Top-1 Test Error (%)	#Params (M)
NASNet-A [57]	26.0	5.3
AmoebaNet-B [31]	27.2	5.3
PNAS [20]	25.8	5.1
DARTS [21]	26.9	4.9
GHN [53]	27.0	6.1
PC-DARTS [48]	24.2	5.3
TE-NAS [2]	24.5	5.4
DI-NAS [28]	25.3	5.2
CATE [49]	25.0	5.8
TA-GATES	24.1	5.6

so they will not become the bottleneck of NAS efficiency. For example, the hypothetical wall time of the search experiment in Table A5 can be estimated as follows: On NB101, the architecture training time of 79 architectures is about 24 GPU hours; On NB201, the architecture training time of 78 architectures is about 32 GPU hours; On NB301, the architecture training time of 58 architectures is about 90 GPU hours.

C.3 Ablation Studies

Ablation study on the iterative encoding process. Table A7 shows the influences of the number of time steps T. Using 2-step TA-GATES is better on NB101, while on NB201 and NB301, using T=3 is better.

About the operation embedding update in Alg. 1 Line 9, we conduct ablation studies on the hyper-parameter s, and whether to mask out the updates for nonparametrized operations. As shown in Table A8, the results are not sensitive to the scale s. As shown in Table A9, on NB101 and NB201, allowing to update non-parametrized operations is beneficial, while on NB301, masking out the updates of non-parametetrized operations is better.

Ablation study on the symmetry-breaking technique. Table A10 shows an ablation study on the hyper-parameter β in Equ. 1. We can see that the results are not sensitive to β.

Table A7: Kendall's Tau of using different number of time steps T.

Step	NB101 (7290 training)				NB201 (7812 training)				NB301 (5896 training)			
	1%	5%	10%	50%	0.5%	1%	5%	10%	0.5%	1%	5%	10%
2	0.6691	**0.7746**	**0.7821**	**0.8114**	**0.6873**	0.7640	0.8604	0.8840	0.5726	0.6345	**0.7138**	0.7302
3	**0.6702**	0.7722	0.7772	0.8103	0.6847	**0.7708**	**0.8632**	**0.8856**	**0.5755**	**0.6363**	0.7133	0.7313
4	0.6572	0.7696	0.7753	0.8082	0.6869	0.7699	0.8621	0.8829	0.5728	0.6351	0.7123	**0.7331**

Table A8: Kendall's Tau of using different update scales s of operation embeddings.

Scale	NB101 (7290 training)				NB201 (7812 training)				NB301 (5896 training)			
	1%	5%	10%	50%	0.5%	1%	5%	10%	0.5%	1%	5%	10%
0.01	0.6648	0.7627	0.7774	0.8087	**0.6858**	0.7641	**0.8646**	0.8824	0.5711	0.6244	0.7169	0.7323
0.1	0.6686	**0.7744**	**0.7839**	**0.8133**	0.6847	**0.7708**	0.8632	0.8856	0.5743	0.6267	0.7184	**0.7328**
1.0	**0.6702**	0.7722	0.7772	0.8103	0.6772	0.7676	0.8638	**0.8871**	**0.5839**	**0.6366**	**0.7189**	0.7312

Table A9: Kendall's Tau of (not) masking out non-parametrized operations.

Mask	NB101 (7290 training)				NB201 (7812 training)				NB301 (5896 training)			
	1%	5%	10%	50%	0.5%	1%	5%	10%	0.5%	1%	5%	10%
w.	0.6654	0.7692	0.7756	0.8050	**0.6881**	0.7642	0.8590	0.8812	**0.5748**	**0.6351**	0.7123	**0.7331**
w/o.	**0.6702**	**0.7722**	**0.7772**	**0.8103**	0.6847	**0.7708**	**0.8632**	**0.8856**	0.5743	0.6267	**0.7184**	0.7328

Table A10: Kendall's Tau of using different scale parameters β in the "Add" symmetry-breaking technique.

Encoder	NB101 (7290 training)				NB201 (7812 training)				NB301 (5896 training)			
	1%	5%	10%	50%	0.5%	1%	5%	10%	0.5%	1%	5%	10%
Add (0.1)	**0.6686**	**0.7744**	**0.7839**	**0.8133**	**0.6753**	0.7720	0.8648	0.8879	**0.5737**	0.6274	0.7079	0.7291
Add (0.5)	0.6677	0.7741	0.7826	0.8120	0.6707	**0.7731**	**0.8660**	**0.8890**	0.5711	0.6335	0.7121	0.7307
Add (1.0)	0.6639	0.7741	0.7822	0.8133	0.6658	0.7649	0.8658	0.8888	0.5728	**0.6351**	**0.7123**	**0.7331**

The symmetry-breaking techniques help provide more discriminative architecture encodings. For example, we train a TA-GATES predictor using the "Concat" symmetry-breaking technique on NB201 (100% training data), and the cosine similarity and L2 distance between the predictor's encodings of Arch-A and Arch-B in Fig. 2 (lower) are 0.88 and 1.46. In contrast, a predictor without symmetry breaking outputs exactly the same encodings for Arch-A and Arch-B.

C.4 Inspection Into Different Time Steps of TA-GATES

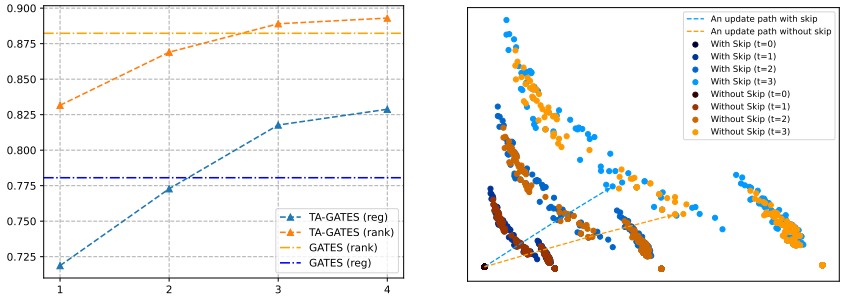
We train two four-step TA-GATES on NB201 with ranking loss and regression loss, respectively. Fig. 9(a) shows how the prediction fitness changes over inference time steps. We can see that the prediction fitness gradually improves as the time step increases during inference and surpasses the baseline GATES in step 3 and 4.

We visualize the updates of the contextualized operation embeddings through time steps in Fig. 9(b). Specifically, we collect 163 architecture pairs from the NB201 test split. All the architectures have a Conv3x3 operation from node 1 to node 2, and the architectures in each pair have only one difference: whether there is a skip connection from node 0 to node 2. We use principal component analysis (PCA) to map operation embeddings of the 1-2 Conv3x3 operations to a 2-dim space, and visualize them. We can see that TA-GATES keeps refining the operation embedding across time steps.

C.5 Inspection Into the Anytime Performance Prediction Task

We show the training curves of TA-GATES and the baseline Multi-GATES in Fig. A10. As shown in Fig. A10(a), we observe a "trade-off" phenomenon of the baseline encoder on NB101, where the KD between the half predicted scores and the half accuracies has an opposite trend with that between the final predicted scores and the final accuracies. We hypothesis that this is because these two supervisory signals contradict each other. In another word, the relative order of half and final accuracies disagree with each other on many pairs of architectures (the KD between the half and final GT accuracy is only 0.4996 on NB101). As a result, directly training baseline encoders with multiple supervisory signals (i.e., the half and final accuracies) even leads to performance degradation compared with Single-GATES.

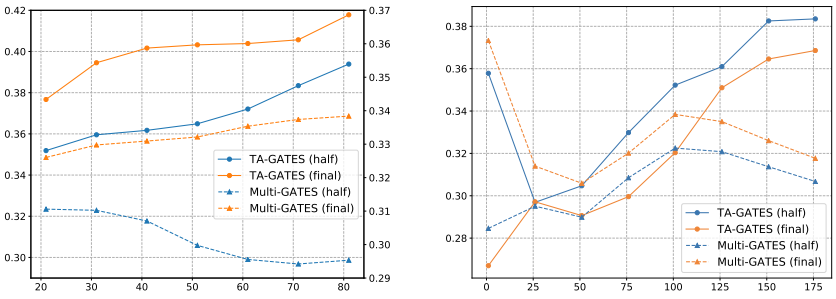
In contrast, TA-GATES eliminates the "trade-off" phenomenon and actually benefits from multiple supervisory signals (as shown in Table 3, Table A12, Table A13, and Table A14). This is because the



(a) Prediction fitness (KD). X-axis: Inference time step; Y-axis: Kendall's Tau.

(b) PCA visualization of Conv3x3 operation embeddings.

Figure A9: The prediction fitness and operation embeddings at different time steps on NAS-Bench 201. The training proportion is 10%.



(a) NB101

(b) NDS-ENAS

Figure A10: The trade-off phenomenon on NB101 and NDS-ENAS. X-axis: Training epochs; Y-axis: Kendall's Tau. The training proportion is 0.5% on NB101 and 5% on NDS ENAS.

analogous modeling of the training process in TA-GATES enables it to capture the learning speed of different architectures to some extent.

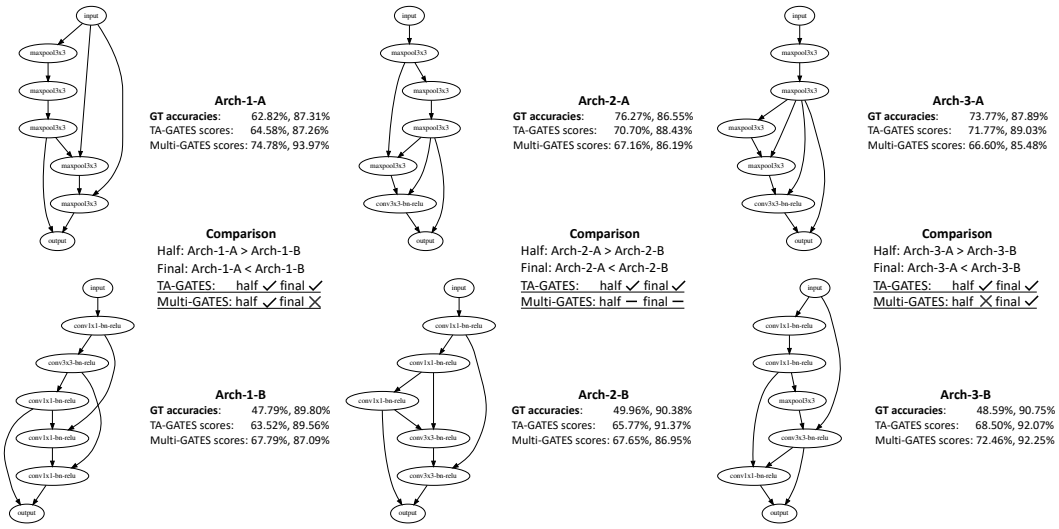


Figure A11: The GT accuracies and predicted scores of some architecture pairs on NB101. TA-GATES can make correct comparisons for the half and final accuracy. Multi-GATES tends to give out the same relative order for the half and final comparisons.

Table A11: Kendall's Tau of anytime training and prediction with sufficient training data. **Upper/Lower**: Kendall's Tau with the half GT accuracy in the middle of training / the final GT accuracy. **Baselines**: "Single-" means to only use one supervisory signal to train a predictor. "Multi-" refers to basic strategy described in Sec. 3.3: The predictor outputs multiple scores and is trained with multiple supervisory signals.

KD with the half accuracy		NB101 (7290 training)		NB301 (5896 training)	
Encoder	Training	50%	100%	50%	100%
Single-GATES	half	0.4796	0.5385	0.7008	0.7896
Multi-LSTM	half+final	0.0518	0.0701	0.3259	0.2797
Multi-GATES	half+final	0.1413	-0.0176	0.2049	0.0637
TA-GATES	half+final	**0.5674**	**0.5538**	**0.7096**	**0.8134**
KD with the final accuracy		NB101 (7290 training)		NB301 (5896 training)	
Encoder	Training	50%	100%	50%	100%
Single-GATES	final	0.5903	0.5875	0.6527	0.7510
Multi-LSTM	half+final	0.1473	0.1468	0.3270	0.2674
Multi-GATES	half+final	0.1818	-0.0213	0.1891	0.0784
TA-GATES	half+final	**0.6477**	**0.6909**	**0.7162**	**0.8107**

To further verify our hypothesis, we show the GT accuracies and predicted scores of some architecture pairs in Fig. A11. Arch-1-A, Arch-2-A, and Arch-3-A with fewer parameters learn at a fast speed and have higher half accuracies, while Arch-1-B, Arch-2-B, Arch-3-B with more parameters learn at a slower speed but have higher final accuracies. We can see that Multi-GATES tends to give out the same relative ranking for the half and final comparisons. Consequently, Multi-GATES gives a wrong final comparison between Arch-1-A and Arch-1-B, and a wrong half comparison between Arch-3-A and Arch-3-B. Multi-GATES also gives out an indistinguishable and wrong comparison between Arch-2-A and Arch-2-B. In contrast, TA-GATES correctly compare the architectures' half and final accuracies in the mean time. This is because the construction of TA-GATES enables it to capture the learning speed characteristics of these architectures to some extent.

As for the ENAS search space, Fig. A10(b) shows that the KDs of baseline predictions with both half and final accuracy drop in the later training period. On the contrary, TA-GATES can obtain constantly increasing KDs during training.

C.6 Additional Results of Anytime Training And Prediction - Regression Measures and the ENAS Search Space

Table A12: Linear correlation (LC) of anytime training and prediction. **Upper/Lower**: LC with the half GT accuracy in the middle of training / the final GT accuracy. **Baselines**: "Single-" means to only use one supervisory signal to train a predictor. "Multi-" refers to basic strategy described in Sec. 3.3: The predictor outputs multiple scores and is trained with multiple supervisory signals.

LC with the half accuracy		NB101 (7290 training)				NB201 (7812 training)				NB301 (5896 training)			
Encoder	Training	1%	5%	10%	50%	0.5%	1%	5%	10%	0.5%	1%	5%	10%
Single-GATES	half	0.4981	0.4815	0.4638	0.7549	0.7802	**0.9016**	0.9627	0.9731	0.3821	0.3868	0.4397	0.4577
Multi-LSTM	half+final	0.0549	0.1458	0.1333	0.1098	0.4113	0.4518	0.6243	0.6786	0.3545	0.3257	0.2906	0.2998
Multi-GATES	half+final	0.4171	0.4224	0.4212	0.1982	0.7701	0.8823	0.9639	0.9739	0.3476	0.3478	0.3348	0.3024
TA-GATES	half+final	**0.5992**	**0.7126**	**0.7375**	**0.7879**	**0.7889**	0.8927	**0.9686**	**0.9758**	**0.4153**	**0.4123**	**0.4649**	**0.4996**
LC with the final accuracy		NB101 (7290 training)				NB201 (7812 training)				NB301 (5896 training)			
Encoder	Training	1%	5%	10%	50%	0.5%	1%	5%	10%	0.5%	1%	5%	10%
Single-GATES	final	0.2725	0.2736	0.3738	0.6090	**0.8848**	0.9515	0.9911	0.9934	0.3872	0.3852	0.4189	0.4330
Multi-LSTM	half+final	0.0318	0.1504	0.2129	0.1517	0.3930	0.4642	0.5765	0.6385	0.3635	**0.4387**	0.4463	0.4580
Multi-GATES	half+final	0.2541	0.2491	0.2491	0.1221	0.7421	**0.9575**	0.9910	0.9951	0.3518	0.3515	0.3390	0.3107
TA-GATES	half+final	**0.5045**	**0.5042**	**0.5397**	**0.6539**	0.7259	0.9570	**0.9952**	**0.9971**	**0.3995**	0.4225	**0.4702**	**0.5025**

Table A13: Mean Squared Error (MSE) of anytime training and prediction. **Upper/Lower**: MSE with the half GT accuracy in the middle of training / the final GT accuracy. **Baselines**: "Single-" means to only use one supervisory signal to train a predictor. "Multi-" refers to basic strategy described in Sec. 3.3: The predictor outputs multiple scores and is trained with multiple supervisory signals. All numbers reported in this table are with a scale of 10^{-2}.

MSE with the half accuracy		NB101 (7290 training)				NB201 (7812 training)				NB301 (5896 training)			
Encoder	Training	1%	5%	10%	50%	0.5%	1%	5%	10%	0.5%	1%	5%	10%
Single-GATES	half	1.8317	1.9187	1.9820	1.1593	**1.2329**	**0.6041**	0.2203	0.1594	0.0635	0.0601	0.0464	0.0340
Multi-LSTM	half+final	2.4588	2.7408	2.4465	2.5831	2.6862	2.7155	1.8496	1.6028	1.9526	1.8609	0.0176	0.0172
Multi-GATES	half+final	1.9623	1.9761	2.1466	2.5757	1.4813	0.7432	0.2092	0.1494	0.0698	0.0655	0.0601	0.0549
TA-GATES	half+final	**1.7778**	**1.2504**	**1.1839**	**0.8783**	1.2714	0.6925	**0.1816**	**0.1376**	**0.0430**	**0.0413**	**0.0313**	**0.0179**
MSE with the final accuracy		NB101 (7290 training)				NB201 (7812 training)				NB301 (5896 training)			
Encoder	Training	1%	5%	10%	50%	0.5%	1%	5%	10%	0.5%	1%	5%	10%
Single-GATES	final	0.8577	0.8553	0.7819	0.6288	1.5868	0.2101	0.0338	0.0260	0.0394	0.0386	0.0320	0.0246
Multi-LSTM	half+final	0.9962	1.0018	0.9543	0.9826	1.5467	1.4989	1.3057	1.0961	3.0224	2.9813	0.0413	0.0075
Multi-GATES	half+final	0.8676	0.8656	0.8741	0.9748	**0.8527**	0.1768	0.0335	0.0190	0.0410	0.0398	0.0353	0.0318
TA-GATES	half+final	**0.5045**	**0.5042**	**0.5397**	**0.6539**	0.9129	**0.1743**	**0.0187**	**0.0110**	**0.0271**	**0.0267**	**0.0197**	**0.0105**

Table A14: Kendall's Tau, Linear Correlation, and Mean Squared Error of anytime training and prediction on NDS ENAS (500 training architectures).

with the half accuracy	Kendall's Tau				Linear Correlation				Mean Squared Error ($\times 10^{-2}$)			
Encoder	5%	10%	50%	100%	5%	10%	50%	100%	5%	10%	50%	100%
Multi-GATES	0.3823	0.3841	0.3554	0.3625	0.1280	0.1300	0.1100	0.0954	**0.8581**	2.0143	1.0979	0.9208
TA-GATES	**0.4257**	**0.4391**	**0.4592**	**0.4755**	**0.1679**	**0.1740**	**0.3426**	**0.4229**	0.8748	**1.1779**	**0.9933**	**0.8123**
with the final accuracy	Kendall's Tau				Linear Correlation				Mean Squared Error ($\times 10^{-2}$)			
Encoder	5%	10%	50%	100%	5%	10%	50%	100%	5%	10%	50%	100%
Multi-GATES	0.3823	0.3841	0.3554	0.3625	0.0773	0.0779	0.0638	0.0502	0.9349	2.3654	**1.0625**	0.9684
TA-GATES	**0.4257**	**0.4391**	**0.4592**	**0.4755**	**0.1027**	**0.1118**	**0.2974**	**0.3754**	**0.9310**	**1.2173**	1.0934	**0.9422**