NASGEM: Neural Architecture Search via Graph Embedding Method

Hsin-Pai Cheng\textsuperscript{1,2} \textsuperscript{*}, Tunhou Zhang\textsuperscript{1} \textsuperscript{†}, Shiyu Li\textsuperscript{1}, Feng Yan\textsuperscript{3}, Meng Li\textsuperscript{2}, Vikas Chandra\textsuperscript{2}, Hai Li\textsuperscript{1}, Yiran Chen\textsuperscript{1}
\textsuperscript{1} Duke University, \textsuperscript{2} Facebook, Inc, \textsuperscript{3} University of Nevada, Reno
\{dave.cheng, tunhou.zhang, shiyu.li, hai.li, yiran.chen\}@duke.edu
\{paicheng, meng.li, vchandra\}@fb.com
\{fyan\}@unr.edu

Abstract

Neural Architecture Search (NAS) automates and prospers the design of neural networks. Recent studies show that mapping the discrete neural architecture search space into a continuous space which is more compact, more representative, and easier to optimize can significantly reduce the exploration cost. However, existing differentiable methods cannot preserve the graph information when projecting a neural architecture into a continuous space, causing inaccuracy and/or reduced representation capability in the mapped space. Moreover, existing methods can explore only a very limited inner-cell search space due to the cell representation limitation or poor scalability. To enable quick search of more sophisticated neural architectures while preserving graph information, we propose NASGEM which stands for Neural Architecture Search via Graph Embedding Method. NASGEM is driven by a novel graph embedding method integrated with similarity estimation to capture the inner-cell information in the discrete space. Thus, NASGEM is able to search a wider space (e.g., 30 nodes in a cell). By precisely estimating the graph distance, NASGEM can efficiently explore a large amount of candidate cells to enable a more flexible cell design while still keeping the search cost low. GEMNet, which is a set of networks discovered by NASGEM, has higher accuracy while less parameters (up to 62\% less) and Multiply-Accumulates (up to 20.7\% less) compared to networks crafted by existing differentiable search methods. Our ablation study on NASBench-101 further validates the effectiveness of the proposed graph embedding method, which is complementary to many existing NAS approaches and can be combined to achieve better performance.

1 Introduction

Neural architecture search (NAS) \cite{1} automates the neural architecture design process. In general, NAS approaches can be divided into the following categories: reinforcement learning based \cite{1}, evolutionary algorithm guided \cite{2}, Bayesian Optimization driven \cite{3}, graph generator based \cite{4}, and differentiable \cite{5, 6, 7, 8} methods. Given the neural architecture design choices are discrete, the approaches in the first four categories search directly over the discrete neural architecture space. However, these methods require to evaluate a large number of candidate neural architectures in a high dimensional discrete sampling space and thus suffer from high search time and computational costs. Gradient and differentiable-based methods like NAO \cite{6}, DARTS \cite{5, 7, 8} were proposed to reduce the high search time and computational costs. Instead of directly searching in the discrete architecture

\textsuperscript{*} Work done while interning at Facebook, Inc
\textsuperscript{†} Equal contribution

Preprint. Under review.
space, differentiable methods map the architecture into a continuous space and optimize the projected continuous space. The advantages are two-fold. First, the projected low-dimensional continuous space is more representative making gradient-based optimization methods easier to be applied; second, optimizing in a continuous space is more efficient because simplified space reduces the redundancy of irrelevant information. Specifically, DARTS [5] projects the optimization objective through encoding the node operations by weighted combination. NAO [6] employs an encoder-decoder structure integrated with an accuracy predictor. Neural architectures are first encoded to a continuous latent space with LSTM cells. The predictor then takes in the continuous representation and predicts its accuracy. Finally the decoder maps the continuous representation back to a neural architecture.

Recent works [4, 9] demonstrate the importance of graph topology in neural architecture design. However, existing differentiable methods [5, 6] cannot precisely preserve graph topology when mapping the architecture into a continuous space. This may result in an inaccurate representation and/or reduced representation capacity of the projected continuous latent space [6]. e.g., the graph distance in the latent space may not appropriately reflect the graph distance in the discrete space and thus the optimal cell found may not be optimal in the graph architecture. In addition, existing differentiable methods do not support flexible inner-cell search. For example, NAO [6] restricts a cell to have two input nodes and one output node. NAO also fixes the number of tensor aggregation in a cell. Such restrictions severely limit the representation capacity of the found cell.

We propose NASGEM (Neural Architecture Search via Graph Embedding Method) to overcome the above limitations of existing differentiable methods. By introducing graph kernel, similarity measure, and efficiency prediction, NASGEM delicately encodes graphs into a latent space and enables to search cells with high representation capacity. Our contributions are as follows: 1) NASGEM preserves the network structure when constructs a graphically meaningful latent space for neural architecture search. 2) NASGEM employs an efficiency score predictor to model the relationship between cell structure and its performance. With the pretrained graph embedding, our predictor can achieve an accurate estimation of the model performance based on the representation of cell structures in a graph vector. 3) The exploration of optimal cell structures is further improved by bootstrap optimization, which guarantees the feasibility of graph vectors in the latent embedding space. 4) GEMNet performs better than models obtained by other differentiable based methods, thanks to the more efficient cell design offered by NASGEM. Specifically, 62% and 8% parameter reduction and 20.7% and 19.3% Multiply-Accumulates (MAC) reduction compared to NAO and DARTS, respectively. Evaluation using NASBench-101 [10] further verifies the effectiveness of graph embedding.

2 Related Work

Graph Embedding. Graph embedding [11, 12] maps topological graph structure into continuous latent space. Traditional vertex graph embedding maps each node to a low-dimensional feature vector meanwhile preserving the connection relationship between vertices. Factorization-based methods [13], random-walk based methods [14], and deep-learning based methods [15] are the popular approaches used in traditional vertex graph embedding. However, it is challenging to apply the existing graph embedding methods to DNNs for the following two reasons: (1) similarities among cell structures of deep neural networks (DNNs) cannot be explicitly derived from traditional graph embeddings; (2) sophisticated deep-learning based methods like DNGR [16] and GCN [17, 18, 19] require complex mechanisms in training structural data. NASGEM addresses cell structure similarity by computing the cosine similarity of two embedded graph vectors. It also facilitates the training process during the formulation of graph embedding by employing an encoding structure.

Neural Architecture Search. Some studies adopt cell-based approach to utilize existing hand-crafted architecture motifs (e.g., MobileNetV2 [20]) and methods that focus on optimization algorithms such as random [21], evolutionary [22, 23], progressive [24], gradient-based [5, 25] and, Bayesian optimization [3, 26], etc. However, as the design space for the cell structures are inherently unchanged, this set of solution for optimal neural architectures still has the capacity limitation. Recent works [6, 5] define a differentiable search space and utilize gradient-based methods to optimize the architecture representation. However, such method usually lacks topological consideration. For example, DARTS [5] and its variations [8, 27] consider only the optimal combination of operations as its optimization goal. Although NAO [6] considers the optimization process within the graph embedding space, strong constraints are applied to the search space. First, NAO uses only 5 nodes
Figure 1: Workflow of NASGEM. (a) Encoder Training: the encoder learns to map graphs into a continuous embedding space. (b) Searching: within the constructed embedding space, NASGEM builds an efficiency score predictor to learn the relationship between neural network topologies and estimated score. (c) Bootstrap Optimization: we utilize the trained efficiency score predictor to identify the best cell topology from a large amount of sampled graphs derived from the supernet.

when constructing the search space for cell structures, which is much smaller compared to 30 nodes in NASGEM. In addition, NAO enforces all existing nodes to establish connections with at most two previous nodes. As a comparison, NASGEM allows the exploration of neural architectures with more freedom. It is worth noting that several searching techniques have been proposed to further improve the differentiable searching such as early stopping, partially channel connection [7, 27]. These improvements are orthogonal to the key design of NASGEM and can be combined with our approach for better performance, which we consider as our future work.

Network Wiring Exploration. Recently, neural architectures with sophisticated wiring superseded traditional chain-like neural architectures with similar computation cost [4]. Random wiring [4] shows that using random graphs as priors when generating neural architectures achieves significant performance boost on various tasks. However, unconstrained random wiring may suffer from the following problems: (1) simply tuning hyperparameters for random prior generation may inevitably constrain the search space which can be otherwise explored (e.g., by NASGEM). This limitation on search space may reduce the quality of cell structures explored by the random wiring. (2) Unconstrained aggregation of tensors leads to high memory consumption within wired neural architectures. NASGEM solves this problem by constraining the maximum number of concatenations within each cell structure during bootstrap optimization.

3 Methodology

3.1 An Overview of NASGEM

NASGEM applies a differentiable method to conduct efficient architecture search based on continuous vectorized representation of graphs. NASGEM represents DNN architectures as a combination of Directed Acyclic Graphs (DAGs) and targets on optimizing the inner cell structures. NASGEM has three main components: graph embedding, efficiency score predictor, and bootstrap optimization.

NASGEM employs graph embedding to project candidate graphs into a latent space. Specifically, we introduce a graph encoder to project the graphs from a discrete space into a continuous latent space. Under the supervision of a graph kernel, our mapping has physical meanings in a topology space, i.e., the distance of graph vectors in the embedding space reflects the similarity of graph structures. To model the relationship between the graph and the performance of its derived DNN architecture, an efficiency score predictor is adopted and trained through example graphs along the search iterations. We employ bootstrap optimization to explore the optimal cell with the efficiency score predictor.

Fig. 1 depicts the 3-step workflow of NASGEM. In the first step, we construct a graph encoder to efficiently derive graph vectors from candidate graphs. The encoder is trained for representing graphs with the supervision of Weisfeiler-Lehman (WL) kernel [28]. A large number of randomly sampled graph pairs from the supernet are adopted for training the graph encoder as shown in Fig. [1](a). In the second step, we utilize the pretrained graph encoder and conduct architecture search to explore the relationship between graph vectors and their corresponding performance on a small proxy dataset. An efficiency score predictor is introduced and updated during the exploration process as shown
in Fig. 1(b). Finally, we obtain the target optimal cell structure by applying bootstrap optimization within a large sample space. The cell structure with the highest score given by the efficiency score predictor is adopted as the optimal cell structure of our NASGEM search as shown in Fig. 1(c).

3.2 Graph Embedding

We adopt graph embedding to model the similarity of different vectorized representations of DNN cell structures. Specifically, we use a graph vector to represent the topological graph structure in a continuous space to represent the node information (i.e., DNN operation) and its neighboring connections (i.e., distribution of tensors).

To connect graph representations in discrete and continuous space, a graph encoder is introduced to map the discrete graph representations (i.e., adjacency matrix) to continuous vectors. To measure the similarity of graphs in discrete topological space, there are various approaches in graph theory, such as WL kernel [28]. For continuous graph embedding space, cosine similarity is widely used to measure the similarity of graph vectors within the range $[0, 1]$. The encoded vectors aim at preserving graph similarity measured by cosine similarity in the continuous space, as shown in Fig. 2. Therefore, we train the graph encoder with the objective of minimizing the difference between WL kernel value in the discrete graph space and the cosine similarity value in the continuous space.

**Weisfeiler-Lehman (WL) Kernel.** We adopt WL kernel [28] as a similarity measure of two input graphs $G_1, G_2$ as follows:

$$ S_g(G_1, G_2) = k_{WL}^h(A_1, A_2). $$

where $A_1, A_2$ are the adjacency matrices for graph $G_1, G_2$. $h$ denotes the iteration times of computing WL kernel. For graphs with $N$ nodes, the WL kernel can be computed with $h$ iterations in $O(h \times N)$. Compared with other graph similarity metrics, WL kernel is able to measure large and complex graphs with relatively low computational complexity.

The procedure of measuring the similarity between two input graphs is as follows. As a start, each node is labeled with its degree. A lookup table is constructed to map each long string label to a unique integer, and the lookup table is initialized by orderly assigning an integer to each distinct label of the labeled graph. The kernel value is obtained by computing the dot product of the corresponding vectors of two graphs. Finally, the kernel value is used as the measurement of the graph similarity.

**Similarity measurement of graph vectors.** We adopt cosine similarity to measure the similarity of the vectorized graph representations (i.e., vector graphs). The cosine similarity measurement on the embedding space is defined as:

$$ S_c(g_1, g_2) = \frac{g_1 \cdot g_2}{||g_1|| \cdot ||g_2||}, $$

where $g_1$ and $g_2$ are vectorized representation of graph $G_1$ and $G_2$. The direction of the vector reflects the proximities of the original graphs. Compared with the standard Euclidean distance in $\mathbb{R}^n$, cosine similarity is scale-invariant. In addition, since the range of cosine distance is bounded, it is comparable with the similarity value given by WL kernel. Thus we can supervise the training of the embedding function with the difference between the cosine similarity value and the WL kernel value.

**Graph encoder.** We employ a graph encoder to map discrete graph structures to a continuous graph vector space. We extract adjacency matrix $A$ from the graph $G$ as the most important feature for mapping, denoted as $A \in \mathbb{R}^{n \times n}$ for a graph with $n$ nodes. The graph encoder learns a mapping function $f : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^d$, which takes in the adjacency matrix $A$ and projects the graph into a $d$-dimension real space. As graph similarities are always measured pairwise, the encoder is trained and evaluated on a large number of graph pairs $(G_1^{(i)}, G_2^{(j)})$. To formulate a meaningful mapping to represent topological graph structure in a continuous space, the cosine similarity of the encoded graph vectors shall represent the similarity of the corresponding original graph in the original discrete space.
Algorithm 1 Training predictor in NASGEM

Input: a supernet $G_s = (V_s, E)$ with $n$ nodes, a pretrained graph encoder function $f^* : \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^d$, an efficiency score predictor $P$, search iterations $T$

begin
  for $i = 1, 2, ..., T$ do
    randomly sample a subgraph $G^{(i)} = (V^{(i)}, E^{(i)})$ from the supernet $G_s$;
    map the current sampled subgraph $G^{(i)}$ to continuous graph vectors by using its adjacency matrix $A$: $g_i = f^*(A)$;
    build candidate DNN architecture from $G^{(i)}$;
    train on proxy dataset and fetch efficiency score: $S(G^{(i)}) = ACC[N(G^{(i)})] - \lambda \log(MAC[N(G^{(i)})])$;
    update the predictor $P$ by using the embedded graph vector $g_i$ and feedback efficiency score $S(G^{(i)})$;
  end for

end

topological space. To satisfy the above training objectives, we define the loss function with respect to a pair of input graph: $(G_1^{(i)}, G_2^{(i)})$, followed by the corresponding adjacency matrix $(A_1^{(i)}, A_2^{(i)})$ as:

$$L(G_1^{(i)}, G_2^{(i)}, f) = [S_e(f(A_1^{(i)}), f(A_2^{(i)})) - S_d(G_1^{(i)}, G_2^{(i)})]^2.$$  

Therefore, the optimal mapping function $f^*$ needs to satisfy the condition below:

$$f^* = \arg \min_f \sum_i L(G_1^{(i)}, G_2^{(i)}, f).$$  

Following most practices in auto-encoders [29], we adopt a fully-connected feedforward network to model the target function $f$. The encoder is trained independently before the searching procedure. We randomly generate a number of graph structure pairs with the same numbers of nodes and apply WL kernel to provide ground truth labels.

3.3 Optimal Cell Exploration

Like most of the previous works [5, 25], we map DAGs to cell structures, which are used as building blocks of DNNs. Each node in the DAG stands for a valid DNN operation (e.g., convolution $1 \times 1$, separable convolution $3 \times 3$, etc.), and each edge stands for the flow of tensors from one node to another. When mapping cell structures to DNN architectures, nodes with no input connections (i.e., zero in-degree) are dropped while nodes with in-degree larger than one is inserted a concatenate operation. The output of building blocks can be constructed from the leaf nodes with zero out-degree. The concatenation of these leaf nodes along the last dimension gives the output feature maps.

Performance evaluation of graph vectors. To evaluate the performance of DNN architectures formed by the corresponding cell structures of graph vectors, we use efficiency score instead of accuracy as our search metric so that both performance and efficiency are taken into considerations. The efficiency score for candidate graph $G$ is formulated as:

$$S(G) = ACC[N(G)] - \lambda \log(MAC[N(G)]),$$

where $N(G)$ is the DNN constructed with the cell represented by $G$. $ACC$ is the validation accuracy on proxy dataset. $MAC$ is the number of Multiply-Add operations of $N(G)$ measured in Millions. $\lambda$ is a penalty coefficient. We use MAC as the penalty term since it can be precisely measured across search iterations. Most compact models [30, 20, 31] have hundred millions of MACs while large models [32] can have billions of MACs. Here we choose $\lambda = 0.1$ to control the MAC penalty in the same magnitude with accuracy for both small and large models. This penalty function can urge the search iteration towards improving the performance of small models or deflating the complexity of large models, and thus strike a balance between complexity and performance.

Efficiency score predictor. The efficiency score predictor $P(\cdot) : \mathbb{R}^d \rightarrow \mathbb{R}$ maps the $d$-dimensional graph vector to a real-value that indicates the performance of DNN architectures built upon this graph vector. For a candidate neural architecture, we use its graph vector as features and the computed efficiency score on the proxy dataset as labels for training the efficiency score predictor.

For simplicity, we adopt a two-layer neural network as the backbone for designing the efficiency score predictor which creates a differentiable function to map architectures to its predictive efficiency score.
The predictor is trained iteratively during the search process. We maintain a set \( \{(g, y)\} \), where \( g \) is a graph vector and \( y \) is its efficiency score measured on proxy dataset. In each iteration, we add current selected graph vector/score pair into the set and train the predictor \( P \) with this enlarged set. Algorithm 1 provides an overview of training the efficiency score predictor.

The predictor becomes more accurate by using the efficiency score of new samples for fine-tuning. More importantly, the predictor in NASGEM is built on top a smoother differentiable space, which is constructed through the graph embedding of unrestricted DAGs. Such accurate prediction allows to explore a wider search space (30 nodes) in extremely small search cost (0.4 GPU days).

**Bootstrap Optimization.** After the predictor is trained using a large number of neural architectures and their corresponding efficiency scores, our goal of finding the optimal cell structure in the topological graph space is equivalent to finding the graph vector that has the highest score according to the efficiency score predictor, formulated as:

\[
g^* = \arg\max_g P(g),
\]

where \( g = f^*(A) \) is the embedded graph vector after passing adjacency matrix \( A \) into the pretrained graph encoder, and \( P(g) : \mathbb{R}^d \rightarrow \mathbb{R} \) is the efficiency score predictor that estimates the efficiency score of a given cell \( A \).

NASGEM explores a continuous immense search space consisting of hyper-complex families of cell structures. For cells with \( N \) nodes, the search space is \( 2^{\frac{N(N+1)}{2}} \), e.g., if \( N = 30 \), there are \( 10^{139} \) possible topologies to explore. For efficient exploration, we introduce an estimation method based on two empirical beliefs: (1) optimal cell structures within the search space is not unique as various architectures of the similar isomorphism can yield equally competitive results. (2) Finding the optimal graph vector in the continuous search space and then decoding may not discover a valid architecture, as the mapping from graph vectors to discrete topological cell structures may not be injective.

**Bootstrap Optimization** addresses the above issues by sampling the cell structure with replacement among a large sample space \( S \) and picking up the best one as our post-searching approximation. With the pretrained graph encoder \( f \), we randomly sample cell structure \( A \in S \) from the sample space and approximate the best candidate cell structure \( A^* \) by predicting the efficiency score using the accuracy predictor \( P \):

\[
A^* = \arg\max_{A \in S} P(f(A)).
\]

Bootstrap optimization serves as a similar function as the decoder in NAO [6]. However, a decoder is not always feasible when mapping the optimal embedding back to the adjacency matrix. Since NASGEM does not rely on decoder, the performance of the optimal cell structure found by NASGEM is sensitive to the size of the sample space. Empirically, a sampling size of 50,000 can sufficiently meet the requirements to get an optimal cell structure.

**4 Experimental Evaluation**

We apply NASGEM to search and evaluate efficient mobile neural architectures for different computation regimes on image classification tasks. We initiate architecture search with a complete DAG of \( N = 30 \) so that a rich source of cells can be constructed. Each node can choose from either \( 1 \times 1 \) convolution or \( 3 \times 3 \) depthwise separable convolution as its candidate DNN operation and each convolution operation adopts a Convolution-BatchNorm-RelU triplet. We use \( 1/10 \) of the CIFAR-10 dataset as the proxy dataset for training the efficiency accuracy predictor. It is worth noting that enabling proxyless [33] setting on ImageNet-1K dataset may yield better cells, but the search cost would be drastically increased.

**4.1 Learning Dynamics of NASGEM**

We illustrate the learning dynamics of NASGEM by plotting the efficiency score surface with respect to latent vectors in the graph embedding space. The efficiency score surface is obtained by sampling 50,000 adjacency matrices, mapping them to the continuous embedding space, and passing them through the efficiency score predictor.

Fig. 3 illustrates the efficiency score surface of efficiency score predictors with and without using graph embedding as a latent vector. Under the guidance of graph kernel embedding, the efficiency
Table 1: ImageNet results with different computation budget. For fair comparison, the input image resolution is fixed at $224 \times 224$. We sort different architectures based on MACs. $N$ denotes the maximum number of DNN node operations explored within a cell structure. Note that the pareto front of reducing MACs and the number of parameters (#Params) is not linear. Further reducing computation cost under a small computation regime is very challenging as redundancy is already low.

| Architecture       | N   | Test Err.($\%$) | #Params (M) | MACs (M) | Search Cost (GPU days) |
|--------------------|-----|-----------------|-------------|----------|------------------------|
| NAONet [6]         | 5   | 25.7 8.2        | 11.35       | 584      | 200                    |
| DARTS [5]          | 5   | 26.7 8.7        | 4.7         | 574      | 4.0                    |
| SNAS [34]          | 5   | 27.3 9.2        | 4.3         | 522      | 1.5                    |
| PC-DARTS [27]      | 5   | 25.1 7.8        | 5.3         | 586      | 0.1                    |
| GDAS [35]          | 5   | 26.0 8.5        | 5.3         | 581      | 0.21                   |
| BayesNAS [26]      | 5   | 26.5 8.9        | 3.9         | /        | 0.2                    |
| NGE [19]           | 5   | 25.3 7.9        | 5.0         | 563      | 0.1                    |
| MiLeNAS [36]       | 5   | 25.4 7.9        | 4.9         | 570      | 0.3                    |
| RandWire-WS [4]    | 32  | 25.3$\pm$0.25  | 7.8$\pm$0.15| 5.6$\pm$0.1| 583$\pm$6.2           |
| GEMNet-A (Ours)    | 30  | 25.3 7.9        | 4.3         | 463      | 0.4                    |
| GEMNet-B (Ours)    | 30  | 24.8 7.8        | 4.5         | 563      | 0.4                    |

score surface is smoother, which allows finding local optimum efficiently. This can be interpreted as the embedding of the optimal cell structure in the continuous embedding space. In contrast, in the case of without graph kernel guided embedding, the efficiency score surface cannot be constructed smoothly, which makes the optimization process more difficult and less efficient.

4.2 ImageNet Results

Table 1 summarizes the key performance metrics on the ImageNet dataset. For a fair comparison, we compare with the most relevant NAS works which adopts a differentiable method under similar search cost and search space (e.g., without predefined MobileNetV2 backbone). Note that our reported search cost includes all phases: encoder training, searching, and bootstrap optimization. Performance of GEMNet-A is 1.4% more accurate than models crafted using another differentiable method, DARTS [5], while having 8% and 19.3% reduction on parameters and MACs, respectively. Compared with NAONet, GEMNet-A is more accurate on top-1 accuracy while 62% and 20.7% reduction on parameters and MACs respectively.

Compared with other NAS models, both GEMNet-A and GEMNet-B have higher accuracy but significantly fewer parameters and MACs. It is worth to note that although NAS models with more MACs have higher accuracy, it is difficult to deploy them on edge and mobile devices due to the
scarce computational resource and energy efficiency consideration. In addition, as GEMNet does not require two input information from previous cells as in DARTS [5] and NAO [6], it is a more preferable option for mobile applications.

4.3 Evaluation on NASBench-101

To facilitate the reproducibility of architecture search and evaluate the selection of search strategy, NASBench-101 [10] is introduced to provide abundant results for various neural architectures (about 432k) in a large number of search spaces. To fairly justify the effectiveness of NASGEM, we further corroborate our searching on NASBench-101. We randomly sample a fixed number of candidate architectures (100∼2000) from NASBench-101 within a fixed operation list. Then, we train the efficiency score predictor with/without our proposed graph embedding on these sampled architecture respectively. Finally, we select the best architecture through bootstrap optimization by using the trained efficiency score predictor to evaluate a total of 50k architectures and pick the best one.

We measure the dispersion between NASGEM and NASBench-101 by Global Prediction Bias, \( B = A_B - A_P \). Here \( A_B \) is the global accuracy, which is the best accuracy in the whole search space given in NASBench-101. \( A_P \) is the predicted accuracy, which is the accuracy achieved by NASGEM. When predicted accuracy is very close to global accuracy, global prediction bias (\( B \)) is moving toward zero. Therefore, the smaller \( B \) reflects the more accurate prediction.

Topo-logy-aware NASGEM search with graph embedding can achieve better prediction of global accuracy than topology-agnostic search methods, as the embedded graph vectors contain extra information from the topological space. As shown in Fig. 4 with the guidance of graph kernel embedding, there is around 0.2% gain on predicting global accuracy on the NASBench-101 dataset. Such topological information can facilitate the training of efficiency score predictor given insufficient data. We also observe the smaller search space in NASBench-101 facilities the training of efficiency score predictor. However, when expanding to a DAG with 30 nodes, only a very small portion of architectures within this search space can be sampled to build the efficiency score predictor, which leads to insufficient examples for methods which utilized raw adjacency matrices. Thus, graph embedding is a very powerful mechanism in boosting the performance of efficiency score predictor. NASGEM also produces more stable results as structural knowledge represented by graph embedding generalizes better than binary adjacency matrices. Thus, the performance of NASGEM is less sensitive to the number of evaluated samples in the search space, leading to a better trade-off between search cost and performance.

5 Conclusion

In this work, we propose NASGEM, a differentiable neural architecture search method via graph embedding method. NASGEM is the first of its kind differentiable NAS method that can search an efficient neural architecture cell topology from an unrestricted wide search space (e.g., 30 nodes in a cell). NASGEM tackles down the limitations of graph topology exploration in existing differentiable search methods by proposing the following ideas: (i) construct a topologically meaningful differentiable space by WL kernel guided graph embedding; (ii) employ an efficiency score predictor to precisely model the relationship between neural architectures and performance; (iii) use bootstrap optimization to explore the optimal cell structure. All these components work coherently to enable NASGEM to search a more efficient neural architecture from an unrestricted wide search space within a short time. Compared with neural architectures produced by existing differentiable methods, GEMNet crafted by NASGEM consistently achieves higher top-1 accuracy while saving up to 62% parameters and 20.7% MACs, respectively. NASGEM is highly adaptable to different search spaces. Providing similar number of candidate nodes in search space as RandWire [4], NASGEM provides
0.5% higher accuracy with 19% reduction on number of parameters. By combining our proposed graph embedding with NASBench-101, it achieves a more precise and stable prediction compared to the version without graph embedding.

References

[1] 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.

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

[3] Kirthevasan Kandasamy, Willie Neiswanger, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Neural architecture search with bayesian optimisation and optimal transport. In *Advances in Neural Information Processing Systems*, pages 2016–2025, 2018.

[4] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. Exploring randomly wired neural networks for image recognition. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[5] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *Proceedings of the International Conference on Learning Representations*, 2019.

[6] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, pages 7827–7838, 2018.

[7] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhengu Li. Darts+: Improved differentiable architecture search with early stopping. arXiv preprint arXiv:1909.06035, 2019.

[8] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1294–1303, 2019.

[9] Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. Discovering neural wirings. arXiv preprint arXiv:1906.00586, 2019.

[10] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-bench-101: Towards reproducible neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 7105–7114, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

[11] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[12] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[13] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[14] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’14, pages 701–710, New York, NY, USA, 2014. ACM.

[15] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016.

[16] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *Thirty-Third AAAI Conference on Artificial Intelligence*, 2016.

[17] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations*, 2016.
[18] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, Advances in Neural Information Processing Systems 29, pages 3844–3852. Curran Associates, Inc., 2016.

[19] Wei Li, Shaogang Gong, and Zhu Xiatian. Neural graph embedding for neural architecture search. In The AAAI Conference on Artificial Intelligence, 2020.

[20] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4510–4520, 2018.

[21] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. arXiv preprint arXiv:1902.07638, 2019.

[22] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 2902–2911. JMLR.org, 2017.

[23] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. In Proceedings of the International Conference on Learning Representations, 2019.

[24] 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 Proceedings of the European Conference on Computer Vision (ECCV), pages 19–34, 2018.

[25] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In International Conference on Machine Learning, pages 4092–4101, 2018.

[26] Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. In ICML, 2019.

[27] 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, 2019.

[28] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. Journal of Machine Learning Research, 12(Sep):2539–2561, 2011.

[29] Geoffrey E Hinton and Richard S Zemel. Autoencoders, minimum description length and helmholtz free energy. In Advances in neural information processing systems, pages 3–10, 1994.

[30] 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. In arXiv preprint arXiv:1704.04861, 2017.

[31] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2820–2828, 2019.

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

[33] Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In Proceedings of the International Conference on Learning Representations, 2019.

[34] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In International Conference on Learning Representations, 2019.

[35] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1761–1770, 2019.
[36] Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. Milenas: Efficient neural architecture search via mixed-level reformulation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2020.