Does Unsupervised Architecture Representation Learning Help Neural Architecture Search?

Shen Yan, Yu Zheng, Wei Ao, Xiao Zeng, Mi Zhang
Michigan State University
{yanshen6,zhengy30,aowei,zengxia6,mizhang}@msu.edu

Abstract

Existing Neural Architecture Search (NAS) methods either encode neural architectures using discrete encodings that do not scale well, or adopt supervised learning-based methods to jointly learn architecture representations and optimize architecture search on such representations which incurs search bias. Despite the widespread use, architecture representations learned in NAS are still poorly understood. We observe that the structural properties of neural architectures are hard to preserve in the latent space if architecture representation learning and search are coupled, resulting in less effective search performance. In this work, we find empirically that pre-training architecture representations using only neural architectures without their accuracies as labels improves the downstream architecture search efficiency. To explain this finding, we visualize how unsupervised architecture representation learning better encourages neural architectures with similar connections and operators to cluster together. This helps map neural architectures with similar performance to the same regions in the latent space and makes the transition of architectures in the latent space relatively smooth, which considerably benefits diverse downstream search strategies.

1 Introduction

Unsupervised representation learning has been successfully used in a wide range of domains including natural language processing [1, 2, 3], computer vision [4, 5], robotic learning [6, 7], and network analysis [8, 9]. Although differing in specific data type, the root cause of such success shared across domains is learning good data representations that are independent of the specific downstream task. In this work, we investigate unsupervised representation learning in the domain of neural architecture search (NAS), and demonstrate how NAS search spaces encoded through unsupervised representation learning could benefit the downstream search strategies.

Standard NAS methods encode the search space with the adjacency matrix and focus on designing different downstream search strategies based on reinforcement learning [10], evolutionary algorithm [11], and Bayesian optimization [12] to perform architecture search in discrete search spaces. Such discrete encoding scheme is a natural choice since neural architectures are discrete. However, the size of the adjacency matrix grows quadratically as search space scales up, making downstream architecture search less efficient in large search spaces [13]. To reduce the search cost, recent NAS methods employ dedicated networks to learn continuous representations of neural architectures and perform architecture search in continuous search spaces [14, 15, 16, 17]. In these methods, architecture representations and downstream search strategies are jointly optimized in a supervised manner, guided by the accuracies of architectures selected by the search strategies. However, these supervised architecture representation learning-based methods are biased towards weight-free operations (e.g., skip connections, max-pooling) which are often preferred in the early stage of the search process, resulting in lower final accuracies [18, 19, 20, 21].
In this work, we propose arch2vec, a simple yet effective neural architecture search method based on unsupervised architecture representation learning. As illustrated in Figure 1, compared to supervised architecture representation learning-based methods, arch2vec circumvents the bias caused by joint optimization through decoupling architecture representation learning and architecture search into two separate processes. To achieve this, arch2vec uses a variational graph isomorphism autoencoder to learn architecture representations using only neural architectures without their accuracies. As such, it injectively captures the local structural information of neural architectures and makes architectures with similar structures (measured by edit distance) cluster better and distribute more smoothly in the latent space, which facilitates the downstream architecture search. We visualize the learned architecture representations in §4.1. It shows that architecture representations learned by arch2vec can better preserve structural similarity of local neighborhoods than its supervised architecture representation learning counterpart. In particular, it is able to capture topology (e.g. skip connections or straight networks) and operation similarity, which helps cluster architectures with similar accuracy.

We follow the NAS best practices checklist [22] to conduct our experiments. We validate the performance of arch2vec on three commonly used NAS search spaces NAS-Bench-101 [23], NAS-Bench-201 [24] and DARTS [15] and two search strategies based on reinforcement learning (RL) and Bayesian optimization (BO). Our results show that, with the same downstream search strategy, arch2vec consistently outperforms its discrete encoding and supervised architecture representation learning counterparts across all three search spaces.

Our contributions are summarized as follows:

- We propose a neural architecture search method based on unsupervised representation learning that decouples architecture representation learning and architecture search.
- We show that compared to supervised architecture representation learning, pre-training architecture representations without using their accuracies is able to better preserve the local structure relationship of neural architectures and helps construct a smoother latent space.
- The pre-trained architecture embeddings considerably benefit the downstream architecture search in terms of efficiency and robustness. This finding is consistent across three search spaces, two search strategies and two datasets, demonstrating the importance of unsupervised architecture representation learning for neural architecture search.

The implementation of arch2vec is available at https://github.com/MSU-MLSys-Lab/arch2vec.
2 Related Work

Unsupervised Representation Learning of Graphs. Our work is closely related to unsupervised representation learning of graphs. In this domain, some methods have been proposed to learn representations using local random walk statistics and matrix factorization-based learning objectives [8, 9, 25, 26]; some methods either reconstruct a graph’s adjacency matrix by predicting edge existence [27, 28] or maximize the mutual information between local node representations and a pooled graph representation [29]. The expressiveness of Graph Neural Networks (GNNs) is studied in [30] in terms of their ability to distinguish any two graphs. It also introduces Graph Isomorphism Networks (GINs), which is proved to be as powerful as the Weisfeiler-Lehman test [31] for graph isomorphism. [32] proposes an asynchronous message passing scheme to encode DAG computations using RNNs. In contrast, we injectively encode architecture structures using GINs, and we show a strong pre-training performance based on its highly expressive aggregation scheme. [33] focuses on network generators that output relational graphs, and the predictive performance highly depends on the structure measures of the relational graphs. In contrast, we encode structural information of neural networks into compact continuous embeddings, and the predictive performance depends on how well the structure is injected into the embeddings.

Regularized Autoencoders. Autoencoders can be seen as energy-based models trained with reconstruction energy [34]. Our goal is to encode neural architectures with similar performance into the same regions of the latent space, and to make the transition of architectures in the latent space relatively smooth. To prevent degenerated mapping where latent space is free of any structure, there is a rich literature on restricting the low-energy area for data points on the manifold [35, 36, 37, 38, 39]. Here we adopt the popular variational autoencoder framework [37, 27] to optimize the variational lower bound w.r.t. the variational parameters, which as we show in our experiments acts as an effective regularization. While [40, 41] use graph VAE for the generative problems, we focus on mapping the finite discrete neural architectures into the continuous latent space regularized by KL-divergence such that each architecture is encoded into a unique area in the latent space.

Neural Architecture Search (NAS). As mentioned in §1, early NAS methods are built upon discrete encodings [42, 43, 12, 44, 45], which face the scalability challenge [46, 47] in large search spaces. To address this challenge, recent NAS methods shift from conducting architecture search in discrete spaces to continuous spaces using different architecture encoders such as SRM [48], MLP [49], LSTM [14] or GCN [50, 51]. However, what lies in common under these methods is that the architecture representation and search direction are jointly optimized by the supervision signal (e.g., accuracies of the selected architectures), which could bias the architecture representation learning and search direction. [52] emphasizes the importance of studying architecture encodings, and we focus on encoding adjacency matrix-based architectures into low-dimensional embeddings in the continuous space. [53] shows that architectures searched without using labels are competitive to their counterparts searched with labels. Different from their approach which performs pretext tasks using image statistics, we use architecture reconstruction objective to preserve the local structure relationship in the latent space.

3 arch2vec

In this section, we describe the details of arch2vec, followed by two downstream architecture search strategies we use in this work.

3.1 Variational Graph Isomorphism Autoencoder

3.1.1 Preliminaries

We restrict our search space to the cell-based architectures. Following the configuration in NAS-Bench-101 [23], each cell is a labeled DAG \( G = (V, E) \), with \( V \) as a set of \( N \) nodes and \( E \) as a set of edges. Each node is associated with a label chosen from a set of \( K \) predefined operations. A natural encoding scheme of cell-based neural architectures is an upper triangular adjacency matrix \( A \in \mathbb{R}^{N \times N} \) and an one-hot operation matrix \( X \in \mathbb{R}^{N \times K} \). This discrete encoding is not unique, as permuting the adjacency matrix \( A \) and the operation matrix \( X \) would lead to the same graph, which is known as graph isomorphism [31].
3.1.2 Encoder

To learn a continuous representation that is invariant to isomorphic graphs, we leverage Graph Isomorphism Networks (GINs) [30] to encode the graph-structured architectures given its better expressiveness. We augment the adjacency matrix as \( \hat{\mathbf{A}} = \mathbf{A} + \mathbf{A}^T \) to transfer original directed graphs into undirected graphs, allowing bi-directional information flow. Similar to [27], the inference model, i.e. the encoding part of the model, is defined as:

\[
q(Z|X, \hat{\mathbf{A}}) = \prod_{i=1}^{N} q(z_i|X, \hat{\mathbf{A}}), \quad \text{with} \quad q(z_i|X, \hat{\mathbf{A}}) = \mathcal{N}(z_i|\mu_i, \text{diag}(\sigma_i^2)).
\]  

(1)

We use the \( L \)-layer GIN to get the node embedding matrix \( \mathbf{H} \):

\[
\mathbf{H}^{(k)} = \text{MLP}^{(k)} \left( \left( 1 + \epsilon^{(k)} \right) \cdot \mathbf{H}^{(k-1)} + \hat{\mathbf{A}} \mathbf{H}^{(k-1)} \right), \quad k = 1, 2, \ldots, L,
\]

(2)

where \( \mathbf{H}^{(0)} = \mathbf{X} \), \( \epsilon \) is a trainable bias, and MLP is a multi-layer perception where each layer is a linear-batchnorm-ReLU triplet. The node embedding matrix \( \mathbf{H}^{(L)} \) is then fed into two fully-connected layers to obtain the mean \( \mu \) and the variance \( \sigma \) of the posterior approximation \( q(Z|X, \hat{\mathbf{A}}) \) in Eq. (1). During the inference, the architecture representation is derived by summing the representation vectors of all the nodes.

3.1.3 Decoder

Our decoder is a generative model aiming at reconstructing \( \hat{\mathbf{A}} \) and \( \mathbf{X} \) from the latent variables \( Z \):

\[
p(\hat{\mathbf{A}}|Z) = \prod_{i=1}^{N} \prod_{j=1}^{N} P(\hat{A}_{ij}|z_i, z_j), \quad \text{with} \quad p(\hat{A}_{ij} = 1|z_i, z_j) = \sigma(z_i^T z_j),
\]

(3)

\[
p(\mathbf{X} = [k_1, \ldots, k_N]^T|Z) = \prod_{i=1}^{N} P(\hat{X}_i = k_i|z_i) = \prod_{i=1}^{N} \text{softmax}(W_o z_i + b_o)_{i,k_i},
\]

(4)

where \( \sigma(\cdot) \) is the sigmoid activation, \( \text{softmax}(\cdot) \) is the softmax activation applied row-wise, and \( k_n \in \{1, 2, \ldots, K\} \) indicates the operation selected from the predefined set of \( K \) operations at the \( n^{th} \) node. \( W_o \) and \( b_o \) are learnable weights and biases of the decoder.

3.2 Training Objective

In practice, our variational graph isomorphism autoencoder consists of a five-layer GIN and a one-layer MLP. The details of the model architecture are described in §4. The dimensionality of the embedding is set to 16. During training, model weights are learned by iteratively maximizing a tractable variational lower bound:

\[
\mathcal{L} = \mathbb{E}_{q(Z|X, \hat{\mathbf{A}})}[\log p(\hat{\mathbf{X}}, \hat{\mathbf{A}}|Z)] - D_{KL}(q(Z|X, \hat{\mathbf{A}})||p(Z)),
\]

(5)

where \( p(\hat{\mathbf{X}}, \hat{\mathbf{A}}|Z) = p(\hat{\mathbf{A}}|Z)p(\hat{\mathbf{X}}|Z) \) as we assume that the adjacency matrix \( \mathbf{A} \) and the operation matrix \( \mathbf{X} \) are conditionally independent given the latent variable \( Z \). The second term \( D_{KL} \) on the right hand side of Eq. (5) denotes the Kullback-Leibler divergence [54] which is used to measure the difference between the posterior distribution \( q(\cdot) \) and the prior distribution \( p(\cdot) \). Here we choose a Gaussian prior \( p(Z) = \prod_i \mathcal{N}(z_i|0, 1) \) due to its simplicity. We use reparameterization trick [37] for training since it can be thought of as injecting noise to the code layer. The random noise injection mechanism has been proved to be effective on the regularization of neural networks [55, 56, 37]. The loss is optimized using mini-batch gradient descent over neural architectures.

3.3 Architecture Search Strategies

We use reinforcement learning (RL) and Bayesian optimization (BO) as two representative search algorithms to evaluate arch2vec on the downstream architecture search.
3.3.1 Reinforcement Learning (RL)

We use REINFORCE \[10\] as our RL-based search strategy as it has been shown to converge better than more advanced RL methods such as PPO \[57\] for neural architecture search. For RL, the pre-trained embeddings are passed to the Policy LSTM to sample the action and obtain the next state (valid architecture embedding) using nearest-neighborhood retrieval based on L2 distance to maximize accuracy as reward. We use a single-layer LSTM as the controller and output a 16-dimensional output as the mean vector to the Gaussian policy with a fixed identity covariance matrix. The controller is optimized using Adam optimizer \[58\] with a learning rate of $1 \times 10^{-2}$. The number of sampled architectures in each episode is set to $16$ and the discount factor is set to $0.8$. The baseline value is set to $0.95$. The maximum estimated wall-clock time for each run is set to $1 \times 10^6$ seconds.

3.3.2 Bayesian Optimization (BO)

We use DNGO \[59\] as our BO-based search strategy. We use a one-layer adaptive basis regression network with hidden dimension 128 to model distributions over functions. It serves as an alternative to Gaussian process in order to avoid cubic scaling \[60\]. We use expected improvement (EI) \[61\] as the acquisition function which is widely used in NAS \[45, 49, 50\]. The best function value of EI is set to $0.95$. During the search process, the pre-trained embeddings are passed to DNGO to select the top-5 architectures in each round of search, which are then added to the pool. The network is retrained for 100 epochs in the next round using the selected architectures in the updated pool. This process is iterated until the maximum estimated wall-clock time is reached.

4 Experimental Results

We validate \textit{arch2vec} on three commonly used NAS search spaces. The details of the hyperparameters we used for searching in each search space are included in Appendix.

**NAS-Bench-101.** NAS-Bench-101 \[23\] is the first rigorous NAS dataset designed for benchmarking NAS methods. It targets the cell-based search space used in many popular NAS methods \[62, 63, 15\] and contains 423,624 unique neural architectures. Each architecture comes with pre-computed validation and test accuracies on CIFAR-10. The cell consists of 7 nodes and can take on any DAG structure from the input to the output with at most 9 edges, with the first node as input and the last node as output. The intermediate nodes can be either $1 \times 1$ convolution, $3 \times 3$ convolution or $3 \times 3$ max pooling. We split the dataset into 90% training and 10% held-out test sets for \textit{arch2vec} pre-training.

**NAS-Bench-201.** Different from NAS-Bench-101, the cell-based search space in NAS-Bench-201 \[24\] is represented as a DAG with nodes representing sum of feature maps and edges associated with operation transforms. Each DAG is generated by 4 nodes and 5 associated operations: $1 \times 1$ convolution, $3 \times 3$ convolution, $3 \times 3$ average pooling, skip connection and zero, resulting in a total of 15,625 unique neural architectures. The training details for each architecture candidate are provided for three datasets: CIFAR-10, CIFAR-100 and ImageNet-16-120 \[64\]. We use the same data split as used in NAS-Bench-101.

**DARTS search space.** The DARTS search space \[15\] is a popular search space for large-scale NAS experiments. The search space consists of two cells: a convolutional cell and a reduction cell, each with six nodes. For each cell, the first two nodes are the outputs from the previous two cells. The next four nodes contain two edges as input, creating a DAG. The network is then constructed by stacking the cells. Following \[63\], we use the same cell for both normal and reduction cell, allowing roughly $10^9$ DAGs without considering graph isomorphism. We randomly sample 600,000 unique architectures in this search space following the mobile setting \[15\]. We use the same data split as used in NAS-Bench-101.

For pre-training, we use a five-layer Graph Isomorphism Network (GIN) with hidden sizes of \{128, 128, 128, 128, 16\} as the encoder and a one-layer MLP with a hidden dimension of 16 as the decoder. The adjacency matrix is preprocessed as an undirected graph to allow bi-directional information flow. After forwarding the inputs to the model, the reconstruction error is minimized using Adam optimizer \[58\] with a learning rate of $1 \times 10^{-3}$. We train the model with batch size 32 and the training loss is able to converge well after 8 epochs on NAS-Bench-101, and 10 epochs on NAS-Bench-201 and DARTS. After training, we extract the architecture embeddings from the encoder for the downstream architecture search.
In the following, we first evaluate the pre-training performance of arch2vec (§4.1) and then the neural architecture search performance based on its pre-trained representations (§4.2).

4.1 Pre-training Performance

Observation (1): We compare arch2vec with two popular baselines GAE [27] and VGAE [27] using three metrics suggested by [32]: 1) Reconstruction Accuracy (reconstruction accuracy of the held-out test set), 2) Validity (how often a random sample from the prior distribution can generate a valid architecture), and 3) Uniqueness (unique architectures out of valid generations). As shown in Table 1, arch2vec outperforms both GAE and VGAE, and achieves the highest reconstruction accuracy, validity, and uniqueness across all the three search spaces. This is because encoding with GINs outperforms GCNs in reconstruction accuracy due to its better neighbor aggregation scheme; the KL term effectively regularizes the mapping from the discrete space to the continuous latent space, leading to better generative performance measured by validity and uniqueness. Given its superior performance, we stick to arch2vec for the remainder of our evaluation.

<table>
<thead>
<tr>
<th>Method</th>
<th>NAS-Bench-101</th>
<th>NAS-Bench-201</th>
<th>DARTS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
<td>Validity</td>
<td>Uniqueness</td>
</tr>
<tr>
<td>GAE [27]</td>
<td>98.75</td>
<td>29.88</td>
<td>99.25</td>
</tr>
<tr>
<td>VGAE [27]</td>
<td>97.45</td>
<td>41.18</td>
<td>99.34</td>
</tr>
<tr>
<td>arch2vec (w/o KL)</td>
<td>100</td>
<td>30.31</td>
<td>99.20</td>
</tr>
<tr>
<td>arch2vec</td>
<td>100</td>
<td>44.97</td>
<td>99.69</td>
</tr>
</tbody>
</table>

Table 1: Reconstruction accuracy, validity, and uniqueness of different GNNs.

Observation (2): We compare arch2vec with its supervised architecture representation learning counterpart on the predictive performance of the latent representations. This metric measures how well the latent representations can predict the performance of the corresponding architectures. Being able to accurately predict the performance of the architectures based on the latent representations makes it easier to search for the high-performance points in the latent space. Specifically, we train a Gaussian Process model with 250 sampled architectures to predict the performance of the other architectures, and report the predictive performance across 10 different seeds. We use RMSE and the Pearson correlation coefficient (Pearson’s r) to evaluate points with test accuracy higher than 0.8. Figure 2 compares the predictive performance between arch2vec and its supervised counterpart on NAS-Bench-101. As shown, arch2vec outperforms its supervised counterpart, indicating arch2vec is able to better capture the local structure relationship of the input space and hence is more informative on guiding the downstream search process.

Observation (3): In Figure 3, we plot the relationship between the L2 distance in the latent space and the edit distance of the corresponding DAGs between two architectures. As shown, for arch2vec, the L2 distance grows monotonically with increasing edit distance. This result indicates that arch2vec is able to preserve the closeness between two architectures measured by edit distance, which potentially benefits the effectiveness of the downstream search. In contrast, such closeness is not well captured by supervised architecture representation learning.

Observation (4): In Figure 4, we visualize the latent spaces of NAS-Bench-101 learned by arch2vec (left) and its supervised counterpart (right) in the 2-dimensional space generated using t-SNE. We overlaid the original colorscale with red (>92% accuracy) and black (<82% accuracy) for highlighting purpose. As shown, for arch2vec, the architecture embeddings span the whole latent space, and architectures with similar accuracies are clustered together. Conducting architecture search on such smooth performance surface is much easier and is hence more efficient. In contrast, for the supervised counterpart, the embeddings are discontinuous in the latent space, and the transition of accuracy is non-smooth. This indicates that joint optimization guided by accuracy cannot injectively encode

1The RMSE and Pearson’s r are: 0.038±0.025 / 0.53±0.09 for the supervised architecture representation learning, and 0.018±0.001 / 0.67±0.02 for arch2vec. A smaller RMSE and a larger Pearson’s r indicates a better predictive performance.
architecture structures. As a result, architecture does not have its unique embedding in the latent space, which makes the task of architecture search more challenging.

Observation (5): To provide a closer look at the learned latent space, Figure 5 visualizes the architecture cells decoded from the latent space of arch2vec (upper) and supervised architecture representation learning (lower). For arch2vec, the adjacent architectures change smoothly and embrace similar connections and operations. This indicates that unsupervised architecture representation learning helps model a smoothly-changing structure surface. As we show in the next section, such smoothness greatly helps the downstream search since architectures with similar performance tend to locate near each other in the latent space instead of locating randomly. In contrast, the supervised counterpart does not group similar connections and operations well and has much higher edit distances between adjacent architectures. This biases the search direction since dependencies between architecture structures are not well captured.
NAS Methods | #Queries | Test Accuracy (%) | Encoding | Search Method
---|---|---|---|---
Random Search [23] | 1000 | 93.54 | Discrete | Random
RL [23] | 1000 | 93.58 | Discrete | REINFORCE
BO [23] | 1000 | 93.72 | Discrete | Bayesian Optimization
RE [23] | 1000 | 93.72 | Discrete | Evolution
NAO [14] | 1000 | 93.74 | Supervised | Gradient Decent
BANANAS [49] | 500 | 94.08 | Supervised | Bayesian Optimization
RL (ours) | 400 | 93.74 | Supervised | REINFORCE
BO (ours) | 400 | 93.79 | Supervised | Bayesian Optimization
arch2vec-RL | 400 | 94.10 | Unsupervised | REINFORCE
arch2vec-BO | 400 | 94.08 | Unsupervised | Bayesian Optimization

Table 2: Comparison of NAS performance between arch2vec and SOTA methods on NAS-Bench-101. It reports the mean performance of 500 independent runs given the number of queried architectures.

4.2 Neural Architecture Search (NAS) Performance

NAS results on NAS-Bench-101. For fair comparison, we reproduced the NAS methods which use the adjacency matrix-based encoding in [23], including Random Search (RS) [66], Regularized Evolution (RE) [44], REINFORCE [10] and BOHB [12]. For supervised architecture representation learning-based methods, the hyperparameters are the same as arch2vec, except that the architecture representation learning and search are jointly optimized. Figure 6 and Table 2 summarize our results.

As shown in Figure 6, BOHB and RE are the two best-performing methods using the adjacency matrix-based encoding. However, they perform slightly worse than supervised architecture representation learning because the high-dimensional input may require more observations for the optimization. In contrast, supervised architecture representation learning focuses on low-dimensional continuous optimization and thus makes the search more efficient. As shown in Figure 6 (left), arch2vec considerably outperforms its supervised counterpart and the adjacency matrix-based encoding after $5 \times 10^4$ wall clock seconds. Figure 6 (right) further shows that arch2vec is able to robustly achieve the lowest final test regret after $1 \times 10^6$ seconds across 500 independent runs.

Table 2 shows the search performance comparison in terms of number of architecture queries. While RL-based search using discrete encoding suffers from the scalability issue, arch2vec encodes architectures into a lower dimensional continuous space and is able to achieve competitive RL-based search performance with only a simple one-layer LSTM controller. For NAO [14], its performance is inferior to arch2vec as it entangles structure reconstruction and accuracy prediction together, which inevitably biases the architecture representation learning.

[https://github.com/automl/nas_benchmarks]
<table>
<thead>
<tr>
<th>NAS Methods</th>
<th>CIFAR-10 validation</th>
<th>CIFAR-10 test</th>
<th>CIFAR-100 validation</th>
<th>CIFAR-100 test</th>
<th>ImageNet-16-120 validation</th>
<th>ImageNet-16-120 test</th>
</tr>
</thead>
<tbody>
<tr>
<td>RE [44]</td>
<td>91.08±0.43</td>
<td>93.84±0.43</td>
<td>73.02±0.46</td>
<td>72.86±0.55</td>
<td>45.78±0.56</td>
<td>45.63±0.64</td>
</tr>
<tr>
<td>RS [66]</td>
<td>90.94±0.38</td>
<td>93.75±0.37</td>
<td>72.17±0.64</td>
<td>72.05±0.77</td>
<td>45.47±0.65</td>
<td>45.33±0.79</td>
</tr>
<tr>
<td>REINFORCE [10]</td>
<td>91.03±0.33</td>
<td>93.82±0.31</td>
<td>72.35±0.63</td>
<td>72.13±0.79</td>
<td>45.58±0.62</td>
<td>45.30±0.86</td>
</tr>
<tr>
<td>BOHB [12]</td>
<td>90.82±0.53</td>
<td>93.61±0.52</td>
<td>72.59±0.82</td>
<td>72.37±0.90</td>
<td>45.44±0.70</td>
<td>45.26±0.83</td>
</tr>
<tr>
<td>arch2vec-RL</td>
<td>91.32±0.42</td>
<td>94.12±0.42</td>
<td>73.13±0.72</td>
<td>73.15±0.78</td>
<td>46.22±0.30</td>
<td>46.16±0.38</td>
</tr>
<tr>
<td>arch2vec-BO</td>
<td>91.41±0.22</td>
<td>94.18±0.24</td>
<td>73.35±0.32</td>
<td>73.37±0.30</td>
<td>46.34±0.18</td>
<td>46.27±0.37</td>
</tr>
</tbody>
</table>

Table 3: The mean and standard deviation of the validation and test accuracy of different algorithms under three datasets in NAS-Bench-201. The results are calculated over 500 independent runs.

<table>
<thead>
<tr>
<th>NAS Methods</th>
<th>CIFAR-10 validation</th>
<th>CIFAR-10 test</th>
<th>CIFAR-100 validation</th>
<th>CIFAR-100 test</th>
<th>ImageNet-16-120 validation</th>
<th>ImageNet-16-120 test</th>
</tr>
</thead>
<tbody>
<tr>
<td>RE [44]</td>
<td>91.08±0.43</td>
<td>93.84±0.43</td>
<td>73.02±0.46</td>
<td>72.86±0.55</td>
<td>45.78±0.56</td>
<td>45.63±0.64</td>
</tr>
<tr>
<td>RS [66]</td>
<td>90.94±0.38</td>
<td>93.75±0.37</td>
<td>72.17±0.64</td>
<td>72.05±0.77</td>
<td>45.47±0.65</td>
<td>45.33±0.79</td>
</tr>
<tr>
<td>REINFORCE [10]</td>
<td>91.03±0.33</td>
<td>93.82±0.31</td>
<td>72.35±0.63</td>
<td>72.13±0.79</td>
<td>45.58±0.62</td>
<td>45.30±0.86</td>
</tr>
<tr>
<td>BOHB [12]</td>
<td>90.82±0.53</td>
<td>93.61±0.52</td>
<td>72.59±0.82</td>
<td>72.37±0.90</td>
<td>45.44±0.70</td>
<td>45.26±0.83</td>
</tr>
<tr>
<td>arch2vec-RL</td>
<td>91.32±0.42</td>
<td>94.12±0.42</td>
<td>73.13±0.72</td>
<td>73.15±0.78</td>
<td>46.22±0.30</td>
<td>46.16±0.38</td>
</tr>
<tr>
<td>arch2vec-BO</td>
<td>91.41±0.22</td>
<td>94.18±0.24</td>
<td>73.35±0.32</td>
<td>73.37±0.30</td>
<td>46.34±0.18</td>
<td>46.27±0.37</td>
</tr>
</tbody>
</table>

Table 4: Comparison with state-of-the-art cell-based NAS methods on DARTS search space using CIFAR-10. The test error is averaged over 5 seeds. Stage 1 shows the GPU days (or number of queries) for model search and Stage 2 shows the GPU days for model evaluation.

### 5 Conclusion

*arch2vec* is a simple yet effective neural architecture search method based on unsupervised architecture representation learning. By learning architecture representations without using their accuracies, it constructs a more smoothly-changing architecture performance surface in the latent space compared to its supervised architecture representation learning counterpart. We have demonstrated its effectiveness on benefiting different downstream search strategies in three NAS search spaces. We suggest that it is desirable to take a closer look at architecture representation learning for neural architecture search. It is also possible that designing neural architecture search method using *arch2vec* with a better search strategy in the continuous space will produce better results.

### 6 Acknowledgement

We would like to thank the anonymous reviewers for their helpful comments. This work was partially supported by NSF Awards CNS-1617627, CNS-1814551, and PFI:BIC-1632051.

[https://github.com/quark0/darts/blob/master/cnn/train.py](https://github.com/quark0/darts/blob/master/cnn/train.py)
**Broader Impact**

In this paper, we challenge the common practice in neural architecture search and ask the question: does unsupervised architecture representation learning help neural architecture search? We approach this question through two sets of experiments: 1) the predictive performance comparison and 2) the neural architecture search efficiency and robustness comparison of the learned architecture representations using supervised and unsupervised learning. In both experiments, we found unsupervised architecture representation learning performs reasonably well. Current NAS methods are typically restricted to some small search blocks such as Inception cell or ResNet block, and most of them perform equally well with enough human expertise under this setup. With the drastically increased computational power, the design of the search space will be more complex and therefore hugely increases the search complexity. In such case, unsupervised architecture representation learning may benefit many downstream applications where the search space contains billions of network architectures, with only a few of them trained with annotated data to obtain the accuracy. Supervised optimization in such large search spaces might be less effective. In the future, we suggest more work to be done to investigate unsupervised neural architecture search with different meaningful pretext tasks on larger search spaces. A better pre-training strategy for neural architectures leveraging graph neural networks seems to be a promising direction, as the unsupervised learning method introduced in our paper has already shown its simplicity and effectiveness.

**References**


