DAG-NAS: EXPLAINABLE NEURAL ARCHITECTURE SEARCH FOR REINFORCEMENT LEARNING VIA SCALAR-LEVEL DAG MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

We present an explainable and effective Neural Architecture Search (NAS) framework for Reinforcement Learning (RL). We model a feed-forward neural network as a Directed Acyclic Graph (DAG) that consists of *scalar-level* operations and their interconnections. We train the model for RL tasks using a differentiable search method, followed by pruning the search outcomes. This process results in a compact neural architecture that achieves high performance and enhances explainability by emphasizing crucial information for solving the RL problem. We apply our NAS framework to the Actor-Critic PPO algorithm, targeting both actor and critic networks. We evaluate its performance across various RL tasks. Extensive experiments demonstrate that our architectures achieve comparable performance with significantly fewer parameters while also enhancing explainability by highlighting key features.

1 INTRODUCTION

Machine learning, a subset of artificial intelligence, has undergone a huge evolution driven by the advent of deep learning. A large number of deep learning methods hinge on effective neural network architectures, which are typically crafted by human experts through a time-consuming and often intuition-based process. Neural Architecture Search (NAS) has made a significant paradigm shift by automating the design of neural network architectures. Since the seminal work of Zoph & Le (2017), extensive research in NAS has systematically explored vast architecture spaces, discovering novel and high-performing network designs across various domains like Natural Language Processing (NLP), Computer Vision (CV) and Reinforcement Learning (RL) Klyuchnikov et al. (2020); Kang et al. (2023); Parker-Holder et al. (2022a). Among these, NAS for RL has been relatively less explored due to its unique challenges, such as fragile RL algorithms, high computational costs, and sensitivity to hyperparameters. There is still much potential to be explored in the context of NAS for RL, pushing the boundaries of what machine learning can achieve White et al. (2023).

Despite its considerable potential, NAS encounters various challenges and limitations. One of the major challenges arises from the need for expertise in designing the search space for NAS, which results in a trade-off between the investment in the search space design and the search complexity. In addition, an overly customized search space aimed at optimizing architecture may hinder the exploration of potentially better architectures and introduce bias into the search process Liu et al. (2018). It has been reported that in such overly constrained search spaces, even random search performs equally well Li & Talwalkar (2020).

Another important challenge is the insufficient explainability of the searched architectures. It is hard to understand how they make their decisions and which information is crucial for the decisions. This issue mirrors a broader challenge in deep learning, where the intricate network structure often results in a black-box model with limited insight into their decision-making process. In deep learning, there have been several studies to improve the model explainability Montavon et al. (2017); Lundberg & Lee (2017), which, however, often necessitate several assumptions or are constrained to specific models. Also in NAS, there have been continuing efforts to enhance the explainability of NAS-generated models Kadra et al. (2023), yet there is still much room for improvement. Moreover, in practical RL problems, the agent has to solve sequential decision-making problems given the input of the system state. This state information often includes a number of measurements from various sensors that may be arbitrarily correlated or uncorrelated. In this case, model explainability is crucial for identifying key features needed to solve the problem, which also contributes to reducing the model size.

In this work, we introduce a novel NAS approach that enhances search flexibility and offers high explanability through high-precision DAG modeling. We decompose a typical feed-forward neural network into scalar-level operations using single-output vertices. By relaxing discrete requirements on the operations and interconnections, we employ a fully-differentiable architecture search. The resulting architectures are then discretized and pruned to achieve a lightweight and high-performing design. We demonstrate the effectiveness of our approach across various RL tasks.

Our contributions can be summarized as follows.

- To the best of our knowledge, we are the first to introduce scalar-level DAGs for modeling neural networks in architecture search. Our method reduces the effort required for designing the search space, enhances search flexibility, and offers high explanability.
- We have developed an effective framework of our DAG-NAS for searching and selecting neural architectures. (i) We extend constraint relaxation to both vertices and edges in DAGs, making a fully-differentiable search process applicable. (ii) We introduce a correlation-based pruning to achieve a compact architecture with high explanability.
- We evaluate our framework across a broad spectrum of RL tasks and demonstrate that the architectures discovered by our framework have significantly fewer parameters while achieving comparable performance.

2 Related Works

Neural Architecture Search (NAS) is a prominent branch in automating the machine learning pipeline, and revolves around three core components: search space, search strategy, and performance estimation strategy Elsken et al. (2019). The search space encompasses the set of architectures that can be explored, the search strategy defines how to explore the search space, and the performance estimation strategy determines how to evaluate interim and candidate results. For example, the seminal work of Zoph & Le (2017) searched a CNN architecture with high classification accuracy, adopting a cell-based search space, RL-based search strategy which is rewarded by validation accuracy.

An important and common challenge of NAS is the search space design. It should be sufficiently large to enclose high-performing architectures, and at the same time, carefully constrained for efficient search with feasible computational complexity. This has led to the development of several structured search spaces with chain, cell, or hierarchy structure White et al. (2023), whose effectiveness heavily depends on tasks. Notable works for the improved search space design include dynamic search space Xia et al. (2022); Ci et al. (2021), unlimited search space Geada & McGough (2022), or space evolution Zhou et al. (2021). However, all these approaches still require a substantial amount of manual design for cell structure or feature size.

Another significant challenge in NAS is the difficulty of explaining the searched architecture. Explanability is particularly crucial for NAS-for-RL, since it can be used to identify important features and lighten the model in practice. There have been several efforts to improve the explainability, such as using Bayesian optimization to identify effective motifs Ru et al. (2021) and employing an evolutionary algorithm to examine input-output relationships Carmichael et al. (2023). Also, there were attempts to model cells using DAG, aiming to enhance the interpretability of architectures Lee et al. (2021); White et al. (2020). All these efforts, however, suffer from insufficient explainability, performance degradation, or additional complexity in the cell design, leaving much room for improvement.

Besides the difficulties of designing the search space and explaining the outcomes, NAS-for-RL has unique challenges related to RL, including task design, learning algorithm selection, hyperparameter configuration, and neural architecture design Parker-Holder et al. (2022b). In this work, we focus on the neural architecture design, which has been less explored with only a few studies Franke et al. (2021); Mysore et al. (2021). A noteworthy related work is Weight Agnostic Neural Network (WANN) Gaier & Ha (2019), which examined the potential of architectures in RL tasks without training their weights. WANN uses an evolutionary search strategy to develop the architectures, growing initial small architectures to large ones. Despite its flexibility in searching architectures, WANN suffers from high computational complexity and often results in complex architectures with a substantial number of parameters.

To address the aforementioned challenges, we develop a novel NAS-for-RL solution that significantly reduces human involvement in the architecture design for RL tasks, and yields explainable, compact neural architectures with comparable performance.

3 PRELIMINARIES

RL is deeply rooted in the mathematical framework of Markov Decision Processes (MDPs), which provide a structured way of modeling an environment. An MDP is formally defined as a tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1]$ is the discount factor. Given state $S_t = s$ at time t, an agent takes action $A_t = a$ and the state transits to state $S_{t+1} = s'$ at time $t + 1$ with probability $T(s, a, s') = \Pr(S_{t+1} = s'|S_t = s, A_t = a)$, and the agent obtains reward R_{t+1}.

The agent learns to make decisions through trial-and-error interactions with the environment. The goal of the RL agent is to find an optimal policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected reward sum $\mathbb{E}[G_t]$ where $G_t = \sum_{k=t}^{\infty} \gamma^{k-t} R_{k+1}$. With the advance of deep learning, neural networks have been employed as an approximate decision-making function, replacing the agent. This technique, known as Deep RL (DRL), can be divided into two categories, value-based and policy-based methods. In the value-based method, neural models estimate the value of states and actions, and in the policy-based method, often referred to as the policy gradient method, they directly yield an action for a given state.

PPO is one of the most popular policy gradient methods in DRL. It utilizes two neural networks: one for the policy (actor) and the other for value estimation (critic). Their weights are adjusted based on the gradients of the following loss function:

$$\mathcal{L}(w) = \mathbb{E}_t[\min(r_t(w) \cdot \hat{A}_t,\ clip(r_t(w), \epsilon) \cdot \hat{A}_t], \tag{1}$$

where w denotes the network weight, $r_t(w) = \frac{\pi_w(a_t|s_t)}{\pi_{w_{old}}(a_t|s_t)}$ is the probability ratio for the sampling with current state s_t and action a_t, $clip(a, b) = \max\{1 - b, \min\{a, 1 + b\}\}$, $\hat{A}_t$ is an estimator of the advantage function defined as $\mathbb{E}[G_t|s_t, a_t] - \mathbb{E}[G_t|s_t]$. PPO maintains a relatively small deviation from the previous policy (w_{old}), ensuring training stability and reducing sensitivity to hyperparameters. PPO has shown remarkable performance in various domains, ranging from video games to robotic control, making it a popular baseline RL algorithm.

4 METHODS

In this section, we explain our method. We model a neural network as a scalar-level DAG and construct the DAG supernetwork by relaxing the discrete constraints. Then we conduct architecture searches using the differentiable method, and discretize the results, obtaining a DDAG.

4.1 SEARCH SPACE OF SCALAR-LEVEL DAG

We consider a multi-layer feed-forward network $f : \mathbb{R}^a \to \mathbb{R}^b$ with input $\mathbf{x} = (x_1, ..., x_a)$ and output $\mathbf{y} = (y_1, ..., y_b)$. It is represented as a graph $G(V, E)$ with the set V of vertices and the set E of directed edges or connections. A vertex collects outputs from incoming vertices and conducts an operation. The network can be divided into l layers, and a vertex belongs to one and only one layer. Let L_i denote the set of vertices in the i-th layer. We assume that an edge can be connected from a vertex in L_i to a vertex in L_j, satisfying $i < j$. Thus, the graph is acyclic with one-directional data flow and has no edge between vertices in the same layer. Let $| \cdot |$ denote the set cardinality. For ease of exposition, we number the vertices following the data flow[1] in the forward path, i.e.,

[1] There is a tie if two vertices are parallel in the data flow, in which case we break the tie arbitrarily.

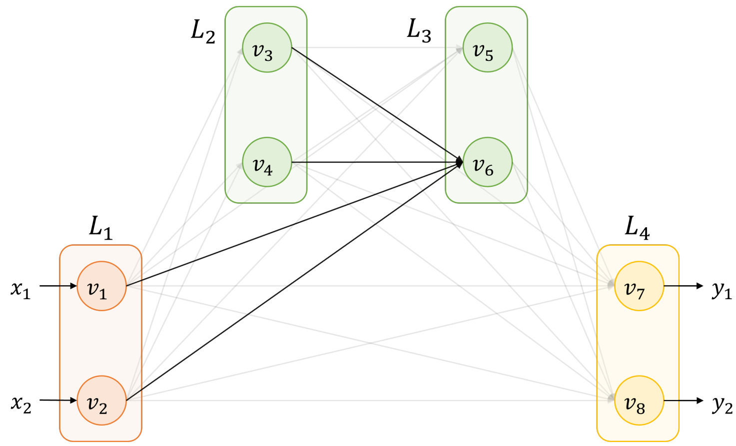

Figure 1: Vertices and connections of DAG for a 4-layer neural network.The possible input connections to vertex v_6 are marked by dark arrows.

$V = \{v_i\}_{i=1}^n$, where v_i is the i-th vertex and $n = |V|$. A directed connection $v_i \to v_j$ is represented by (v_i, v_j), which must satisfy $i < j$ to align with the direction of data flow.

Note that DAG is not sufficient to identify a neural network model, and the same DAG can represent multiple models. To this end, we decompose a neural network model into its architecture, which is represented by a DAG, and weight parameters. Let c_{ij} denote the connectivity between vertices i and j, i.e., $c_{ij} = 1$ if $(v_i, v_j) \in E$, and $c_{ij} = 0$ otherwise. Given input $\mathbf{x}$, the output of v_j can be written as

$$v_j(\mathbf{x}) = o_j \left(\textstyle\sum_{i=1}^{j-1} c_{ij} v_i(\mathbf{x}); \mathbf{w}_j \right), \tag{2}$$

where $o_j(\cdot; \mathbf{w}_j) : \mathbb{R} \to \mathbb{R}$ denotes the operation of vertex v_j, e.g., $LeakyReLU$. We will provide a more detailed description of the operations later. By *model architecture* or architecture, we refer to $\alpha = (\mathbf{o}, \mathbf{c})$ with $\mathbf{o} = \{o_i\}$ and $\mathbf{c} = \{c_{ij}\}$, and by *weight parameters* or weights, we refer to $\mathbf{w} = \{\mathbf{w}_i\}$. Certainly, this modeling can represent any feed-forward neural network, and given the architecture $\mathbf{o}, \mathbf{c}$ and weights $\mathbf{w}$, we can construct a neural network model. We emphasize that each vertex output is a scalar value, and connectivity is defined on a per-vertex basis rather than a per-layer basis. We refer to this structured DAG as a *scalar-level DAG*.

Fig. 1 illustrates a 4-layer neural network with 2 vertices in each layer. Vertex $v_6 \in L_3$ can accept any outputs of vertices in L_1 and L_2, which are marked by dark arrows. This structure admits the representation of a general feed-forward network model with all possible skip connections. Let $\mathbf{h}_i^{out}$ denote the outputs of the vertices in L_i, i.e., $\mathbf{h}_i^{out} = \{v_i(\mathbf{x})\}_{i \in L_i}$, and let $\mathbf{h}_j$ as their concatenations up to $j-1$, satisfying $\mathbf{h}_j = concat(\mathbf{h}_{j-1}, \mathbf{h}_{j-1}^{out})$ and[2] $\mathbf{h}_1 = \mathbf{x}$. The input of vertex operation o_j at layer L_k can be represented as

$$\textstyle\sum_{i=1}^{j-1} c_{ij} v_i(\mathbf{x}) = \langle c_j, h_{k-1} \rangle, \tag{3}$$

where $\mathbf{c}_j = \{c_{ij}\}_{i=1}^{j-1}$ and $\langle \cdot, \cdot \rangle$ denote the inner product. Algorithm 1 describes the forward computation of an l-layer neural network, represented by architecture α and weights $\mathbf{w}$.

In this work, we assume that the number l of layers and the number of vertices at each layer are given, and focus on the search for the operation $\mathbf{o}$ and connectivity $\mathbf{c}$. Extension to the search for l and the number of vertices remains an interesting and important open problem.

4.2 Architecture Search of Scalar-Level DAG

For the architecture search, we employ the well-known differential architecture search of DARTS Liu et al. (2019). It admits gradient-based optimizations by relaxing the discrete architectural con-

[2]The input layer L_1 is an exception, where the vertices are predetermined. Specifically, a vertex in L_1 accepts an element of input data $\mathbf{x}$ and passes itself as the output with no operation, yielding $\mathbf{h}_1 = \mathbf{x}$.

Algorithm 1 Forward computation of neural network.

Input: $\mathbf{x} = (x_1, ..., x_a) \in \mathbb{R}^a$
Parameter: $\alpha = (\mathbf{o}, \mathbf{c}), \mathbf{w}$
Output: $\mathbf{y} = (y_1, ..., y_b) \in \mathbb{R}^b$
$\mathbf{h}_1 \leftarrow \mathbf{x}$
for each $i = 2, \ldots, l$ **do**
  $\mathbf{h}_i^{out} \leftarrow \{\}$
  for each vertex $v_j \in L_i$ **do**
    $\mathbf{h}_i^{out} \leftarrow concat(\mathbf{h}_i^{out}, o_j(\langle \mathbf{c}_j, \mathbf{h}_{i-1} \rangle)$
    $\mathbf{h}_i^{out} \leftarrow concat(\mathbf{h}_i^{out}, o_j(\langle \mathbf{c}_j, \mathbf{h}_{i-1} \rangle; \mathbf{w}_j))$
  end for
  $\mathbf{h}_{i+1} \leftarrow concat(\mathbf{h}_i, \mathbf{h}_i^{out})$
end for
return $\mathbf{y} \leftarrow \mathbf{h}_l^{out}$ =0

straints and enables efficient exploration over a larger search space, becoming one of the most popular search strategies.

For the differentiable search, we replace binary connectivity $c \in \{0, 1\}$ with mixed connectivity $\bar{c} \in [0, 1]$, and denote vertex j's incoming mixed connectivity as $\bar{\mathbf{c}}_j = \{\bar{c}_{ij}\}_{v_i \in L_{<j}}$, where $L_{<j} = \cup_{i<j} L_i$. Similarly, we relax the discrete constraint of vertex operation as follows. Suppose that we have the operation search space $\mathcal{O}$ that consists of $|\mathcal{O}|$ operations, i.e., $\mathcal{O} = \{o^1, \ldots, o^{|\mathcal{O}|}\}$. For each vertex v_j, we introduce operation weights $\mathbf{a}_j = \{a_j^1, \ldots, a_j^{|\mathcal{O}|}\}$ that satisfy $\sum_{k=1}^{|\mathcal{O}|} a_j^k = 1$ and $a_j^k \geq 0$ for all k, and define its mixed operation $\bar{o}_j$ as

$$\bar{o}_j(x) = \textstyle\sum_{k=1}^{|\mathcal{O}|} a_j^k o^k(x). \tag{4}$$

In this work, we consider three candidate operations of $\mathcal{O} = \{Tanh, LeakyReLU, Linear\}$ for each vertex, where $Tanh(x) = tanh(x)$, $LeakyReLU(x) = \max(0.2x, x)$, and $Linear(x) = w_1 x + w_2$. Note that $Linear$ has two trainable weights w_1 and w_2, while $Tanh$ and $LeakyReLU$ have no weight. The set $\mathcal{O}$ can be readily expanded to include additional options.

Through the above relaxation, we obtain a DAG supernetwork with mixed operations and connectivities. This allows us to apply the fully differential architecture search for scalar-level DAG. During the learning procedure, we take the gradients of the loss function, and obtain optimal mixed variables $\bar{\alpha} = (\bar{\mathbf{o}}, \bar{\mathbf{c}})$. The details are as follows.

- We adopt the PPO algorithm for the learning, in which case the PPO objective equation 1 is used as the loss function.
- We take the single-level optimization approach to learn the architecture (α) and weights (w) together:
$$(\alpha^*, w^*) = \arg\min_{\alpha, w} \mathcal{L}(\alpha, w). \tag{5}$$
The single-level optimization is computationally efficient due to fewer optimizations, thus used in NAS works with diverse optimization complexities Xie et al. (2019); An & Joo (2024). Due to the high complexity and instability of learning in RL tasks, reducing the number of optimization steps while matching the direction of optimization is important, and single-level optimization is advantageous in both perspectives.
- After the learning process completes, we store the final architecture $\bar{\alpha}^*$ and weights $\mathbf{w}^*$. Note that, unlike many previous NAS schemes, we do not store intermittent architectures and make use of only the last architecture. This implies that we remove the high-cost after-search evaluation process that most NAS schemes require to determine the final outcome among a number of candidate architectures.

The outcome $\bar{\alpha}^*$ is a supernetwork architecture with mixed variables, which need to be further discretized to obtain a feasible architecture. Fig. 2-(a) shows an example of a DAG supernetwork after the differentiable architecture search, where the relative importance of operations and connectivities in mixed variables is represented by the darkness of the fonts and arrows, respectively.

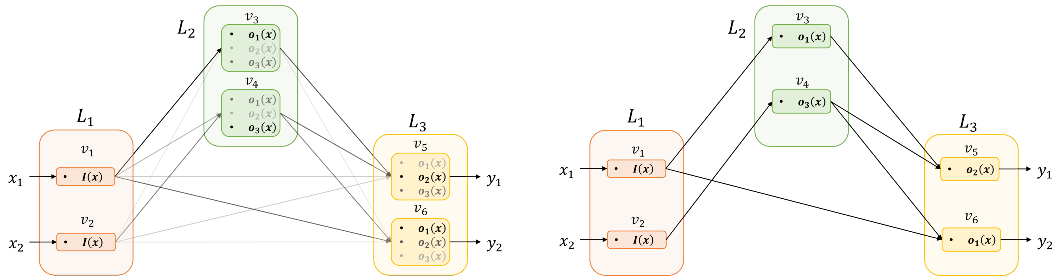

Figure 2: After the differentiable architecture search, we obtain (a) a DAG supernetwork with mixed operations and connectivities. We represent the relative importance of operations and connections by the darkness of fonts and arrows, respectively. From the DAG supernetwork, we obtain the final architecture called (b) Discrete DAG (DDAG) through the discretization process that prunes less important operations and connections.

Remark: The PPO algorithm may fail to achieve high rewards Chan et al. (2020); Agarwal et al. (2021), in which case, a single search results in a low-performing architecture. In our work, we repeat the search 5 times and select the best outcome.

4.3 Architecture Discretization

After the search, we have a fully trained DAG supernetwork with learned connectivities and operation importances. We obtain a discrete architecture by choosing one operation out of the candidate operations in each vertex, and by discretizing the connectivities. This is the process of finalizing architecture $\alpha^* = (\mathbf{o}^*, \mathbf{c}^*)$ from the search outcome $\bar{\alpha}^* = (\bar{\mathbf{o}}^*, \bar{\mathbf{c}}^*)$.

For vertex v_j, we select its operation with the largest importance, i.e.,

$$o_j^* = o^\kappa, \text{ where } \kappa = \arg\max_k a_j^k. \tag{6}$$

If the chosen operation includes learnable weights (e.g., $Linear$), we also use the weights from $\mathbf{w}^*$ without retraining.

For connectivity, we discretize the mixed connectivities based on the correlation between the vertex outputs as follows.

1. We generate a number of synthetic inputs $\mathbf{x}$ by sampling uniformly from $[-1, 1]^a$, where a is the input dimension.
2. We forward the synthetic inputs and compute the mutual correlations between the vertex outputs *within the same layer*. We partition[3] the vertices such that all the vertices in the same group have a mutual correlation higher than 0.9. In each group, we maintain one vertex that is selected arbitrarily and remove all the other vertices from the supernetwork.
3. Then, we compute the correlation between vertices i, j *across layers* and discretize the connectivity using a threshold of 0.5, i.e., $c_{ij}^* = 1$ if the correlation is greater than 0.5 and $c_{ij}^* = 0$ otherwise.

After the discretization process, we obtain a Discrete DAG architecture, or simply DDAG, as shown in Fig. 2-(b). Once DDAG is determined, we keep the trained weights $\mathbf{w}$ from the supernetwork and omit the typical weight-retraining process of NAS. DDAG has significantly fewer parameters while still achieving good performance.

[3]For the same setting, it is possible to group the vertices differently. We arbitrarily select one.

5 EXPERIMENTS

We apply our DAG-NAS framework to the well-known RL environments in the Gymnasium with actor-critic PPO agents. The learning of the PPO agent and the architecture search of DAG-NAS occur simultaneously. We consider a three-layer DAG supernetwork architecture for similar representation power as typical MLPs. The results show that the search outcomes of DAG-NAS achieve comparable performance in most RL tasks, witnessing its effectiveness. Further, the outcome architectures clarify which input elements are of importance to solve the RL problems, demonstrating better explainability.

Our implementation of actor-critic style PPO follows CleanRL Huang et al. (2022). We slightly modify it by separating the actor and critic networks, both of which are a simple 2-layer[4] architecture with $nn.Linear$ in PyTorch library. They have $a \times b + b$ and $a \times 1 + 1$ learnable weights, respectively, where a and b are the input and output dimensions, respectively. This will serve as the baseline architecture throughout our experiments. Accordingly, we design the DAG supernetwork such that the number of parameters of DDAG does not exceed that of the baseline. We train the weights of the baseline for 10 million steps. Also, we train DAG supernetworks for the same number of steps. Note that the training of DAG supernetworks involves both architecture and weight training under the single-level optimization equation 5. On completion of the training, we fix the weights of the architecture outcome and test it across 100 episodes, yielding 100 final rewards. The final score will be their average. During the training, we do not set a seed to control the randomness. Instead, we repeat the training 5 times (trials) and select the one with the highest final score. Throughout the entire procedure involving multiple searches and evaluations, we consistently observed a stable final architecture and evaluation results, demonstrating its robust behavior.

Our experiments were conducted across a total of 17 RL environments, which include Classic Control (Acrobot, CartPole, MountainCar, MountainCarContinuous, Pendulum), Box2D (LunarLander, LunarLanderContinuous, BipedalWalker, BipedalWalkerHardcore), and MUJOCO (Pusher, Reacher, Hopper, Ant, HalfCheetah, InvertedDoublePendulum, InvertedPendulum, Walker2d).

In the following, we compare the performance of baseline, DAG supernetwork, and DDAG (i.e., the outcome of DAG-NAS). We then discuss the explainability of the search outcomes and the impact of architecture on sample efficiency. Details regarding environmental versions, hyperparameters, reward values, and diagrams of the searched architectures are available in the supplementary material.

5.1 PERFORMANCE EVALUATION

We evaluate our method by comparing the performance of the baseline architecture, DAG supernet obtained after the differentiable search, and DDAG obtained after discretization. For each pair of PPO actor-critic networks, we evaluate their performance over 100 episodes and collect the corresponding rewards.

Fig. 3 presents the evaluation results in terms of the achieved rewards and the number of parameters of the searched architectures. Specifically, across 17 RL environments (x-axis), it illustrates the interquartile range (IQR) of the rewards using min-max normalized boxplots (top figure), and the number of the parameters using log-scaled bars (bottom figure).

In classic control and Box2D environments, all three achieve comparable performance, with the DAG supernet substantially outperforming the others in certain environments, such as CartPole, BipedalWalker, and BipedalWalkerHardcore. In MUJOCO environments with continuous robotic control, the DAG supernet and DDAG show comparable performance in most cases, except for Hopper and HalfCheetah. First, we note that DDAG outperforms the baseline in most instances, while having ten times fewer parameters. Second, the DAG supernet may suffer from high computational complexity due to its significantly larger number of parameters. Third, despite the large number of parameters, the DAG supernet may not always outperform the others. For example, DDAG surpasses the DAG supernet in the Pendulum environment.

In summary, our DAG-NAS discovers DDAGs that achieve performance comparable to their baseline counterparts in most environments, while being up to ten times smaller in size.

[4] Including the input layer.

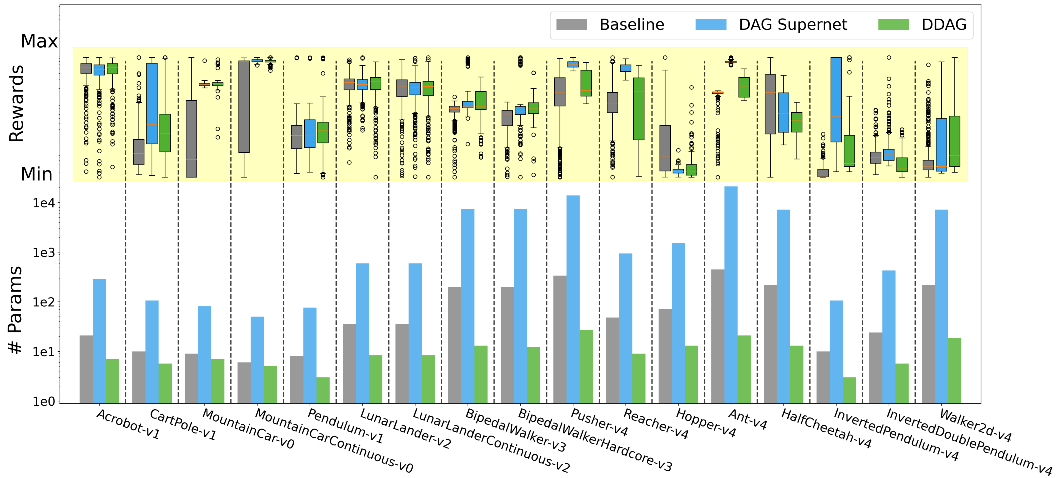

Figure 3: Performance comparison of baseline architectures and the architectures found by DAG-NAS (DAG supernets and DDAGs) in 17 RL environments. We attempt 100 trials in each environment and show the interquartile range (IQR) of achieved (normalized) rewards by boxplots (top figure). Also, the number of the architecture parameters is presented in a log scale (bottom figure). Overall DDAGs perform well with a much smaller number of parameters, and in some cases, even outperform DAG supernets.

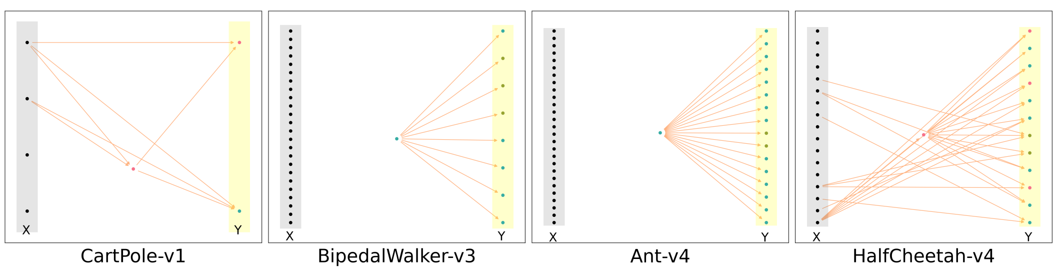

Figure 4: DDAG actor architectures. The left vertices (shaded in gray) and right vertices (shaded in yellow) represent the input and output layers, respectively.

5.2 Architecture Analysis

In this section, we closely examine the searched architectures, as shown in Fig. 4, which displays the actor networks of DDAG outcomes in the CarPole (Classic Control), BipedalWalker (Box2D), Ant, and InvertedPendulum (MUJOCO) environments. In each figure, the data flow from the left input $\mathbf{x} \in \mathbb{R}^a$ to the right output $\mathbf{y} \in \mathbb{R}^b$, with the input and output[5] dimensions vary for each environment. There are three columns of dots, each dot representing a vertex in the three layers, and the arrows between dots denote the connections. The sizes of the search space differ according to the environment, ranging from 2.38×10^7 (MountainCarContinuous) to 5.619×10^{2457} (Ant).

In the Cartpole environment, the state consists of [Cart Position, Cart Velocity, Pole Angle, Pole Angular Velocity]. However, the DDAG actor network utilizes only the last two inputs to produce an action, as shown in Fig. 4. Similarly, only a portion of the state inputs are used in the DDAG actor networks in other classic control environments. This implies that feature selection is integrated into the training of the DAG supernet and becomes evident during the discretization process.

[5]For the environments with continuous action space, the output represents means $y_1, \ldots, y_{b/2}$ and standard deviations $y_{b/2+1}, \ldots, y_b$ of possible actions. The action of dimension $\mathbb{R}^{b/2}$ will be obtained by sampling from a normal distribution.

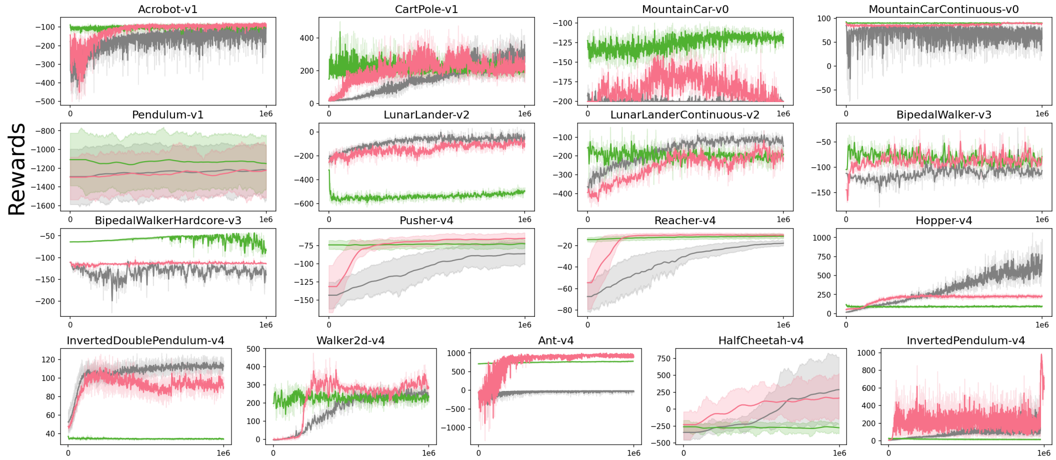

Figure 5: Reward traces for 1 million steps for DDAG (green), DDAG-RW (red), and the baseline (gray).

For the Box2D environments, LunarLander and LunarLanderContinuous are identical except for their output formats: LunarLander has 4-dimensional discrete actions that execute one of thrusting left/right/ground-oriented engines or doing nothing, and LunarLanderContinuous has 2-dimensional continuous actions for the left/right-oriented engines and the ground-oriented engine. Interestingly, the searched DDAG actors for both environments are very similar, indicating that DAG-NAS can successfully identify the essential features to solve RL problems. Similar results are observed in the DDAG actors for BipedalWalker and BipedalWalkerHardcore, which have the same state and action spaces but different terrains.

Fig. 4 also presents the DDAG actors for Ant and HalfCheetah environments in MUJOCO. Notably, the Ant actor does not utilize any state information to produce an action. Similar architectures that do not utilize input are found in BipedalWalker, Pusher, Reacher, and Walker2d. These environments have continuous action spaces, and the agent samples an action from the means and standard deviation outputs. All these DDAG actors output a constant value for the means and standard deviations, implying that the challenges presented in these environments are relatively straightforward, allowing high-reward actions through the learning of constant values. In contrast, the DDAG actors searched in Hopper, HalfCheetah, InvertedPendulum, and InvertedDoublePendulum make use of some state information.

The connectivities in DDAG highlight valuable information, enhancing the model's explainability. While the vertex operations have a relatively small impact on performance, this impact might vary with the complexity of RL problems. In situations where deep, intricate features are necessary, vertex operations could become crucial. Additionally, our framework demonstrates consistency, yielding nearly identical architectures across multiple trials.

5.3 Architecture Impact on Sample Efficiency

In the previous sections, we set the weights $\mathbf{w}$ of DDAG to be the same as those of the DAG supernet, allowing us to bypass the typical weight-retraining process of NAS and directly evaluate the neural network model $(\alpha^*, \mathbf{w}^*)$. In this section, we further focus on architectural superiority. To this end, we initialize the weights of DDAG at random and then retrain them, which is denoted by DDAG-RW. By doing this, we aim to demonstrate that DDAG excels not only in achieving high rewards but also in rapidly acquiring knowledge, highlighting its significant contribution to sample efficiency.

We compare the training performance of the baseline, DDAG, and DDAG-RW over 1 million steps. We conduct 5 trials for each experiment and report the mean and standard error of rewards with a rolling window of 20. Fig. 5 shows their learning curves in 17 environments. Typically, DDAG starts strong and maintains its performance throughout training, but sometimes its performance declines as training continues. In contrast, DDAG-RW initially exhibits lower performance yet eventually matches or even surpasses DDAG. Compared to the baseline, DDAG-RW attains higher rewards in fewer steps, likely due to effective feature selection and fewer parameters. However, there are a few cases, such as Hopper, where baseline outperforms both DDAG and DDAG-RW. We observe that their corresponding DAG supernet also underperforms the baseline, indicating an unsuccessful architecture search.

6 CONCLUSION

We develop a fully differentiable NAS framework for RL through scalar-DAG modeling of neural networks. We simplify cell designs and structure their connections, and successfully relax the discrete constraints, creating a DAG Supernetwork. We then develop a correlation-based pruning method that produces a Discrete DAG (DDAG) architecture with significantly fewer parameters. Additionally, we eliminate the conventional weight-retraining step in NAS, making the architecture search process more practical.

Testing across various RL environments, we demonstrate the effectiveness and flexibility of DAG-NAS. The derived DDAGs achieve high rewards despite their lightweight nature and are self-explainable through the connections on important features. We also highlight the architectural superiority of DDAGs in terms of sample efficiency. Notably, retrained DDAGs exhibit accelerated learning compared to the baselines.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021.

Taegun An and Changhee Joo. Cycleganas: Differentiable neural architecture search for cyclegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 1655–1664, June 2024.

Zachariah J Carmichael, Tim Moon, and Sam Ade Jacobs. Learning debuggable models through multi-objective NAS. In *AutoML Conference 2023*, 2023.

Stephanie C.Y. Chan, Samuel Fishman, Anoop Korattikara, John Canny, and Sergio Guadarrama. Measuring the reliability of reinforcement learning algorithms. In *International Conference on Learning Representations*, 2020.

Yuanzheng Ci, Chen Lin, Ming Sun, Boyu Chen, Hongwen Zhang, and Wanli Ouyang. Evolving search space for neural architecture search, 2021.

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

Jörg K.H. Franke, Gregor Koehler, André Biedenkapp, and Frank Hutter. Sample-efficient automated deep reinforcement learning. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=hSjxQ3B7GWq`.

Adam Gaier and David Ha. Weight agnostic neural networks, 2019.

Rob Geada and Andrew Stephen McGough. Spidernet: Hybrid differentiable-evolutionary architecture search via train-free metrics. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 1962–1970, June 2022.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

Arlind Kadra, Sebastian Pineda Arango, and Josif Grabocka. Breaking the paradox of explainable deep learning, 2023.

Jeon-Seong Kang, JinKyu Kang, Jung-Jun Kim, Kwang-Woo Jeon, Hyun-Joon Chung, and Byung-Hoon Park. Neural architecture search survey: A computer vision perspective. *Sensors*, 23(3), 2023. ISSN 1424-8220.

Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, and Evgeny Burnaev. Nas-bench-nlp: Neural architecture search benchmark for natural language processing, 2020.

Hayeon Lee, Eunyoung Hyung, and Sung Ju Hwang. Rapid neural architecture search by learning to generate graphs from datasets. In *International Conference on Learning Representations*, 2021.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In Ryan P. Adams and Vibhav Gogate (eds.), *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pp. 367–377. PMLR, 22–25 Jul 2020.

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)*, September 2018.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211–222, 2017. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2016.11.008.

Siddharth Mysore, Bassel El Mabsout, Renato Mancuso, and Kate Saenko. Honey. i shrunk the actor: A case study on preserving performance with smaller actors in actor-critic rl. In *2021 IEEE Conference on Games (CoG)*. IEEE Press, 2021.

Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, Frank Hutter, and Marius Lindauer. Automated reinforcement learning (autorl): A survey and open problems. *Journal of Artificial Intelligence Research*, 74:517–568, June 2022a. ISSN 1076-9757. doi: 10.1613/jair.1.13596. URL `http://dx.doi.org/10.1613/jair.1.13596`.

Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, Frank Hutter, and Marius Lindauer. Automated reinforcement learning (autorl): A survey and open problems. *Journal of Artificial Intelligence Research*, 74:517–568, June 2022b. ISSN 1076-9757. doi: 10.1613/jair.1.13596. URL `http://dx.doi.org/10.1613/jair.1.13596`.

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*, 2021.

Colin White, Willie Neiswanger, Sam Nolen, and Yash Savani. A study on encodings for neural architecture search. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 20309–20319. Curran Associates, Inc., 2020.

Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers, 2023.

Xin Xia, Xuefeng Xiao, Xing Wang, and Min Zheng. Progressive automatic design of search space for one-shot neural architecture search. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 2455–2464, January 2022.

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

Daquan Zhou, Xiaojie Jin, Xiaochen Lian, Linjie Yang, Yujing Xue, Qibin Hou, and Jiashi Feng. Autospace: Neural architecture search with less human interference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 337–346, October 2021.

Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

A Appendix

A.1 Hyperparameters

We slightly modified the reliable PPO implementation of CleanRL Huang et al. (2022). We use the PPO parameters shown in Table 1. The same hyperparameters are used for the baseline, DAG supernet, and DDAG, across all the 17 environments.

Table 1: Hyperparameters of PPO algorithm.

Name	notation	value
gamma	γ	0.99
GAE Lambda	λ	0.95
Value function coefficient	c_{vf}	0.5
Entropy coefficient	c_{ent}	0
Clipping coefficient	c_{vf}	0.2
Normalized advantage	adv_norm	True
Clip value loss	cvl	True
Target KL divergence	KL_{target}	None
max grad norm	c_{vf}	0.5
update$_epochs$	K	4
rollout$_steps$	T	512
num minibatches	mB	8
num envs	-	4
num steps	-	1000000
Learning rate	-	0.0002
Learning rate annealing	-	False
beta 1	β_1	0.9
beta 2	β_2	0.999

A.2 Rewards

The rewards of Fig.3 in the main paper are min-max normalized. Table 2 shows the mean and standard deviation of unnormalized rewards.

A.3 DDAG Architectures Searched by DAG-NAS

In this section, we report the architectures of actor and critic networks, found in 17 environments. The left subfigure shows the architecture of the actor network and the right subfigure shows the architecture of the critic network. In each figure, we enlist the input features on the left side using the notation $x_1, ..., x_a$ and output features on the right side as $y_1, ..., y_b$. Note that in continuous control tasks, $y_1, ..., y_{b/2}$ is for the mean value of actions, and $y_{b/2+1}, ..., y_b$ is for the standard deviation of the actions. We present the vertex with the discrete operation of $Tanh$ as pink, $LeakyReLU$ as olive green, and $Linear$ as mint.

Table 2: Mean and standard deviation of rewards, obtained from 100 trials in each environment.

Environment	DAG	DDAG	Baseline
Acrobot-v1	-89.52 ± 34.50	-86.96 ± 26.64	-87.17 ± 27.02
CartPole-v1	261.98 ± 166.36	193.68 ± 132.65	125.44 ± 82.17
MountainCar-v0	-111.49 ± 2.79	-111.34 ± 8.51	-167.33 ± 34.33
MountainCarContinuous-v0	91.45 ± 1.47	90.85 ± 2.13	49.46 ± 51.89
Pendulum-v1	-1241.07 ± 277.03	-1231.57 ± 300.03	-1284.92 ± 273.85
LunarLander-v2	-179.92 ± 129.61	-188.18 ± 208.19	-185.98 ± 188.34
LunarLanderContinuous-v2	-207.2 ± 188.34	-210.22 ± 211.84	-205.78 ± 204.88
BipedalWalker-v3	-101.77 ± 31.65	-105.0 ± 26.61	-123.24 ± 22.28
BipedalWalkerHardcore-v3	-96.06 ± 43.17	-99.82 ± 23.01	-126.86 ± 25.73
Pusher-v4	-55.25 ± 7.73	-104.29 ± 32.91	-135.57 ± 73.18
Reacher-v4	-11.48 ± 2.23	-31.25 ± 18.50	-31.15 ± 12.90
Hopper-v4	73.85 ± 38.8	131.05 ±171.37	353.88 ± 334.47
Ant-v4	955.53 ± 42.11	148.08 ±391.87	-127.15 ± 334.73
HalfCheetah-v4	146.91 ± 330.88	-25.17 ± 215.17	323.22 ± 521.49
InvertedDoublePendulum-v4	576.25 ± 354.48	279.68 ± 295.53	39.06 ± 50.6
InvertedPendulum-v4	110.0 ± 39.82	67.72 ± 33.22	93.36 ± 34.27
Walker2d-v4	113.01 ± 136.32	134.53 ±124.1	55.3 ± 88.82

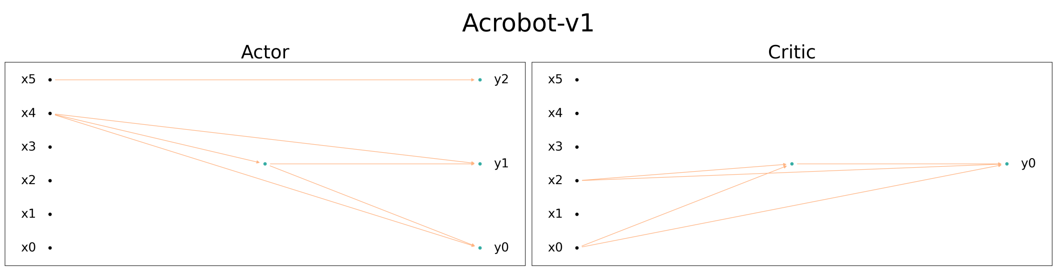

Figure 6: Architectures of actor and critic networks for the Acrobot-v1 environment.

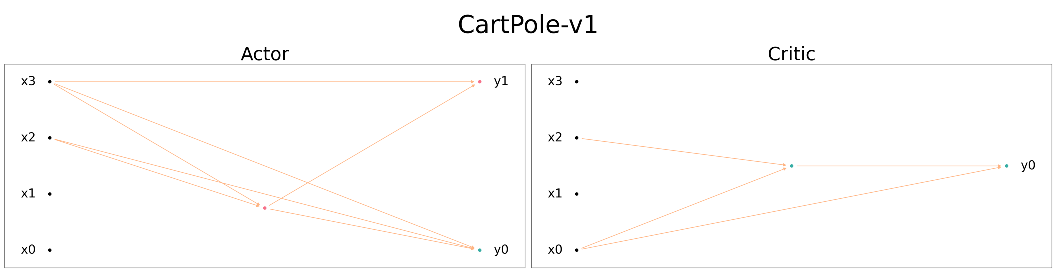

Figure 7: Architectures of actor and critic networks for the CartPole-v1 environment.

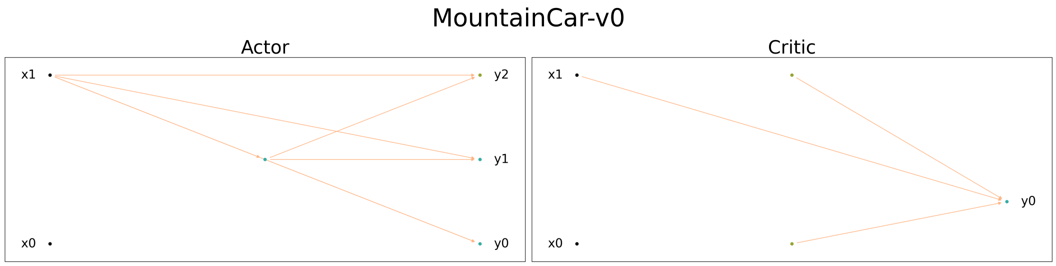

Figure 8: Architectures of actor and critic networks for the MountainCar-v1 environment.

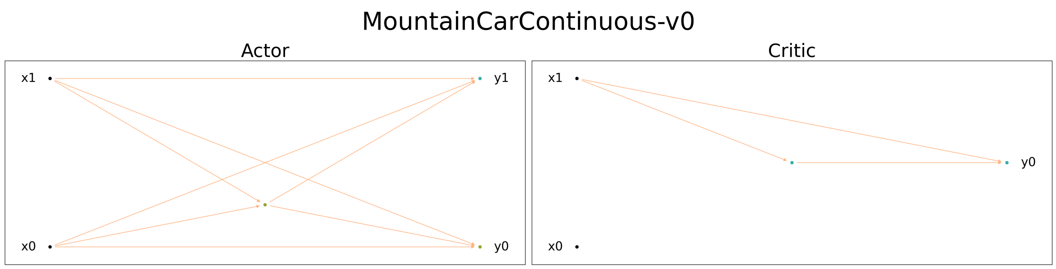

Figure 9: Architectures of actor and critic networks for the MountainCarContinuous-v1 environment.

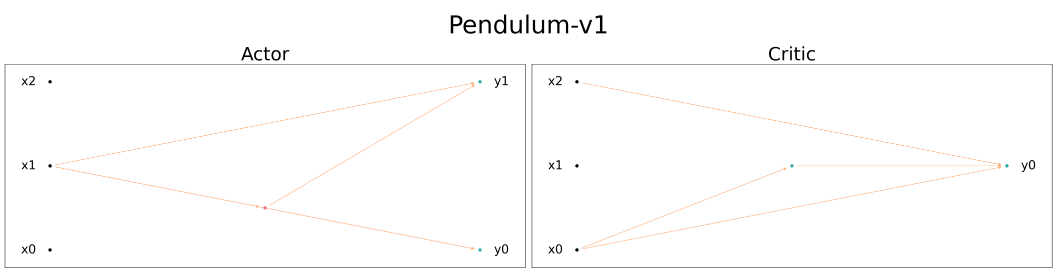

Figure 10: Architectures of actor and critic networks for the Pendulum-v1 environment.

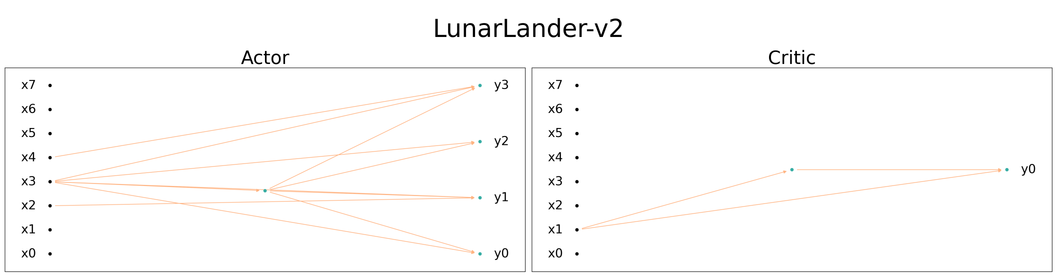

Figure 11: Architectures of actor and critic networks for the LunarLander-v2 environment.

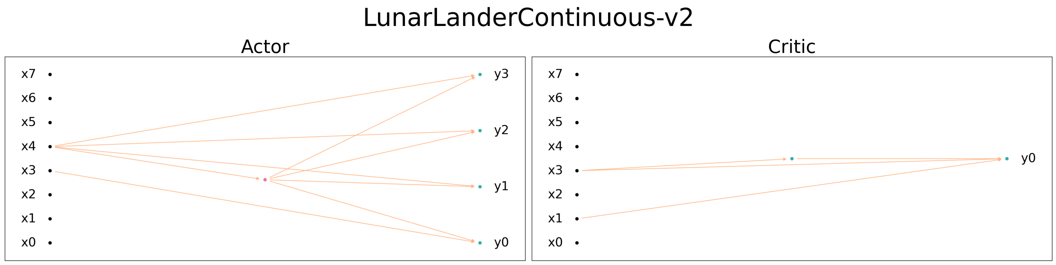

Figure 12: Architectures of actor and critic networks for the LunarLanderContinuous-v2 environment.

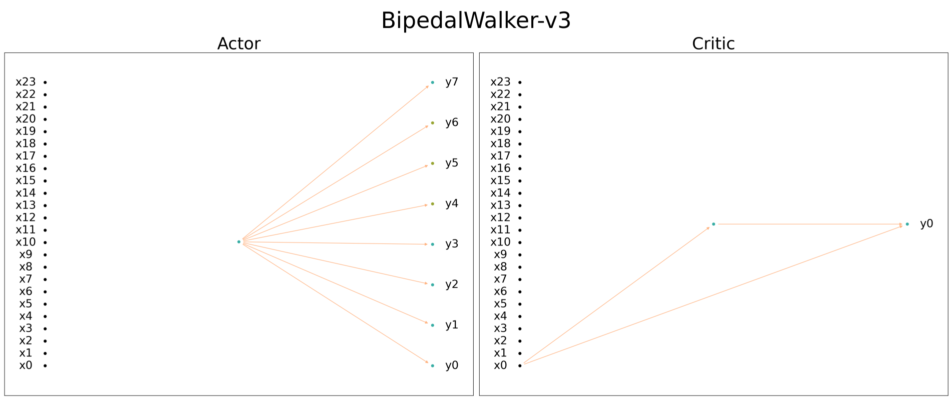

Figure 13: Architectures of actor and critic networks for the BipedalWalker-v3 environment.

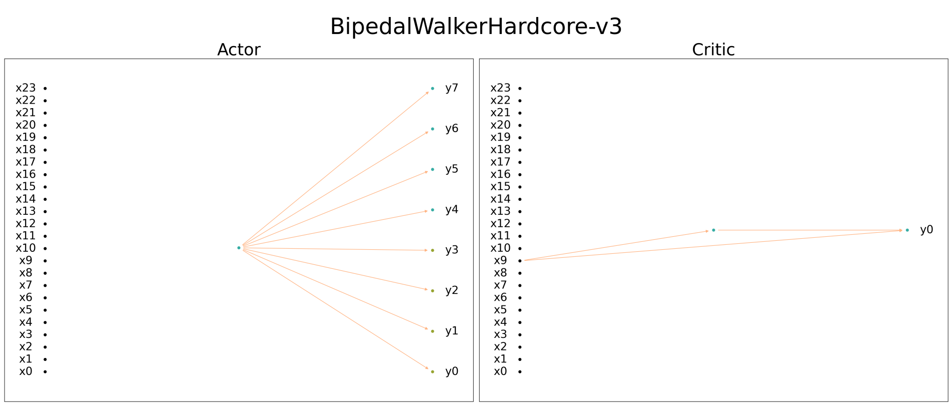

Figure 14: Architectures of actor and critic networks for the BipedalWalkerHardcore-v4 environment.

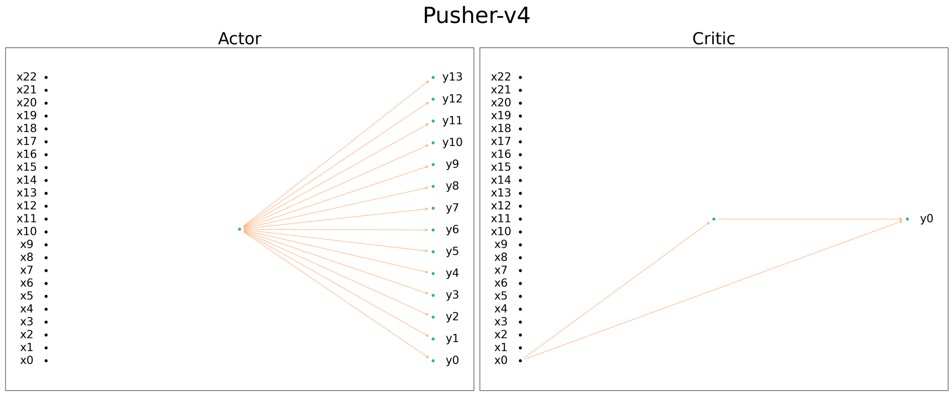

Figure 15: Architectures of actor and critic networks for the Pusher-v4 environment.

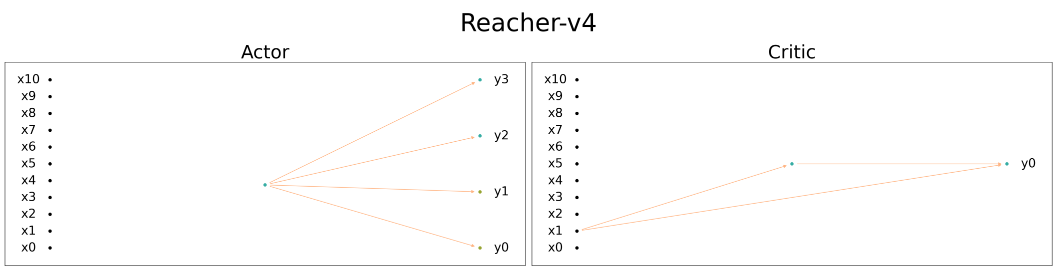

Figure 16: Architectures of actor and critic networks for the Reacher-v4 environment.

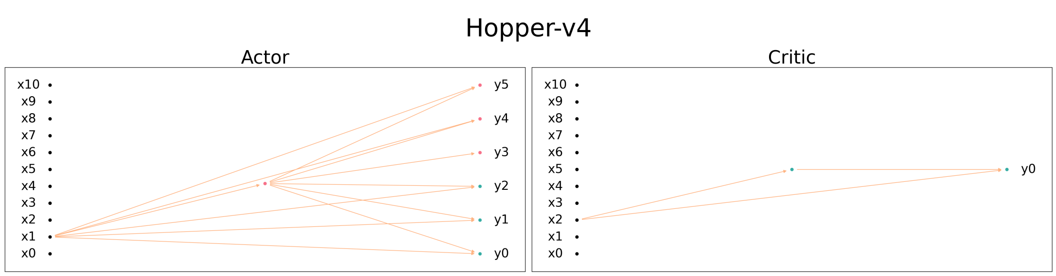

Figure 17: Architectures of actor and critic networks for the Hopper-v4 environment.

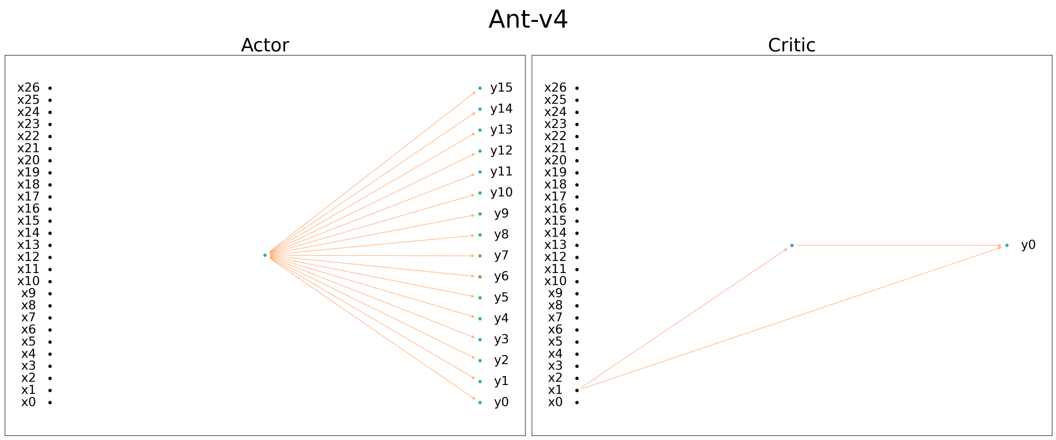

Figure 18: Architectures of actor and critic networks for the Ant-v4 environment.

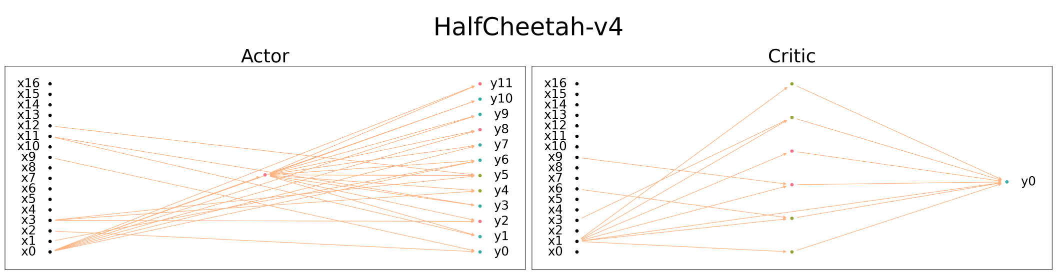

Figure 19: Architectures of actor and critic networks for the HalfCheetah-v4 environment.

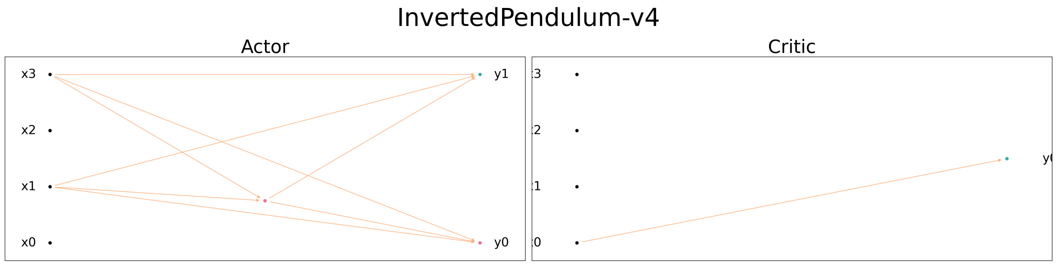

Figure 20: Architectures of actor and critic networks for the InvertedPendulum-v4 environment.

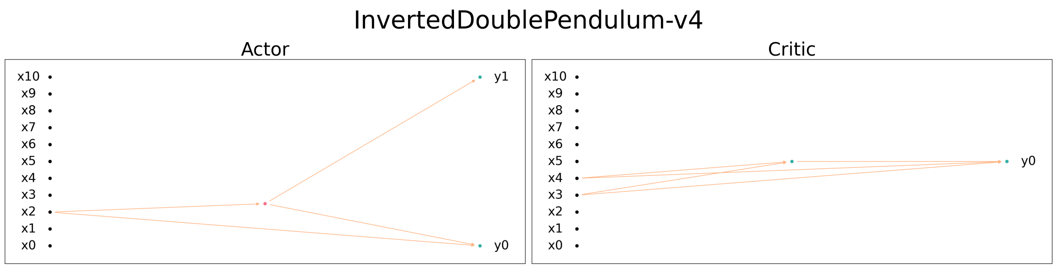

Figure 21: Architectures of actor and critic networks for the InvertedDoublePendulum-v4 environment.

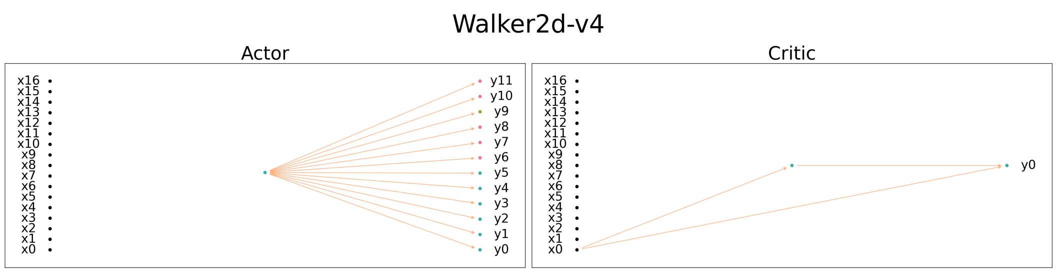

Figure 22: Architectures of actor and critic networks for the Walker2d-v4 environment.

A.4 DAG-NAS WITH DEEPER NETWORKS

In addition, we present the result of DAG-NAS with deeper networks on Hopper-v4. We construct DDAG with 5 layers including an input layer where each layer has a configuration of $[11, 88, 176, 352, 6]$, and name it as DAG-L. We trained DAG-L for 1M steps, produced DDAG-L, and presented its architecture in Fig. 23. Also, we compare the number of parameters and performances in Fig. 24. The result shows that our DAG framework can be extended to deeper and larger networks.

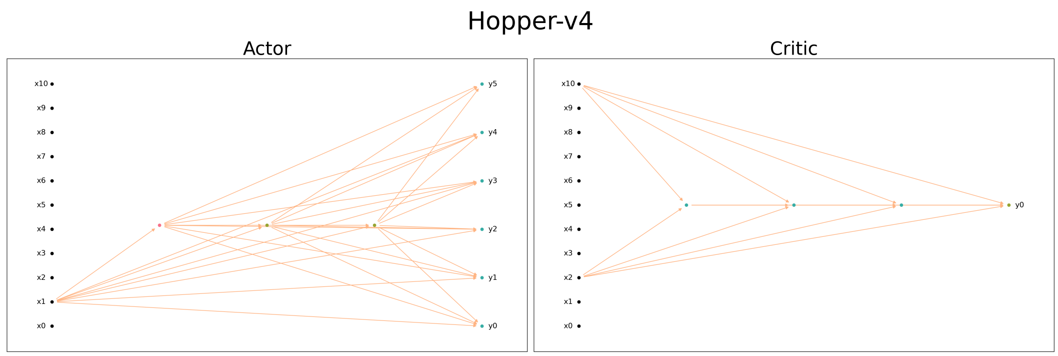

Figure 23: Architectures of DDAG-L actor and critic networks for the Hopper-v4 environment.

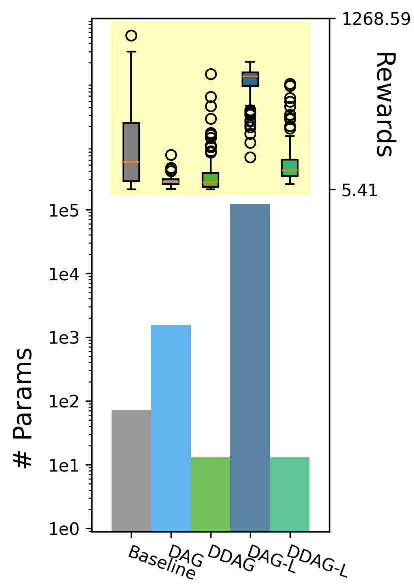

Figure 24: Rewards from 100 episodes in the Hopper-v4 environment.