Neural Architecture Search

Ashish Verma
Department of Data Science
Old Dominion University
Norfolk, VA 23509
averm004@odu.edu

Abstract

The idea behind neural architecture search algorithm is to implemented a recurrent network for generating model descriptions of neural networks.We train the recurrent network using reinforcement learning to maximize expected accuracy on a validation set.This document covers the effectiveness of the approach on the CIFAR-10 multi label classification dataset.Starting from scratch, the method designed a novel network architecture comparable to the best human-invented architecture in terms of test set accuracy.The lower test error rate of 3.65 on the CIFAR-10 dataset is achieved.The model generated from neural architecture search outperformed the previous state-of-the-art model with a similar architectural scheme by 0.09 percent in accuracy and 1.05x in speed.This document is inspired from actual NAS paper which was present in ICRL in 2017 by Google brain team Barret Zoph, Quoc V. Le.

1 Introduction

This document will cover in depth understanding of the neural architecture search various terminologies used throughout the document and we also try to find the answer of the basic question like what is neural architecture search? why do we really need neural architecture search? what are the various advantages and dis advantages of the neural architecture search?.This document also explanation of the various type of neural architecture search.What are the various behind the scenarios and processes involved in NAS? We also try get the basic understanding of the NAS algorithm with the help of reinforcement learning, policy gradient and reward functions.During the end of the document we will talk about various techniques used to accelerate the NAS process and various optimization techniques to quick processing.

2 Terminologies

Recurrent Neural Network: also known as RNNs, are a class of neural networks that allow previous outputs to be used as inputs while having hidden states.

Reinforcement Learning:,RL is a type of machine learning technique that enables an agent to learn in an interactive environment by trial and error using feedback from its own actions and experiences.

Test Error Rate: Test Error rate is expressed as a ratio and is calculated by dividing the total number of pointers read by the total number of errors made in test dataset.

Meta Learning: Meta-learning is a sub-field of machine learning where the focus is on training models to be good learners across a variety of tasks or domains. The idea is to design algorithms or models that can adapt quickly to new tasks with minimal data. Example transfer learning.

Meta Architecture: The meta-architecture defines the general structure of a model that is capable of meta-learning. This might include components for learning how to learn, adaptability to different

Preprint. Under review.

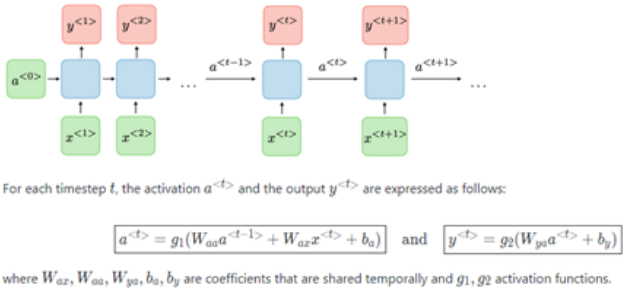


Figure 1: A Generic RNN block

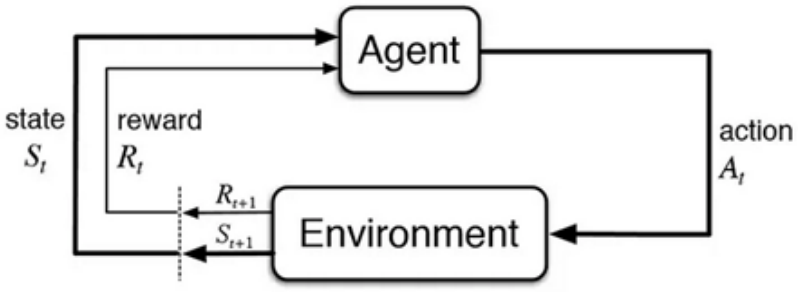


Figure 2: Reinforcement Learning Area

tasks, and mechanisms for transferring knowledge from one task to another. Meta-architectures often include modules or components that facilitate meta-learning. For example, there might be an outer loop that updates the model's parameters based on its performance across different tasks, and an inner loop that fine-tunes the model on individual tasks.

Weight Sharing: Weight sharing is a technique used in neural networks where the same set of weights is reused across different parts of the model. Instead of having separate parameters for each part or unit of the network, weight sharing involves sharing the same weights among multiple units or layers.

Network Morphism: Network morphism refers to a concept in neural network design and optimization where a neural network is transformed or morphed into another network with a similar architecture but different parameters. The idea behind network morphism is to maintain the network's overall structure while allowing for structural changes that can improve performance, adapt to new tasks, or optimize for specific criteria.

Policy Gradient: Policy gradient is a technique used in reinforcement learning to optimize the policy of an agent. In reinforcement learning, an agent learns to make decisions by interacting with an environment. The policy is a strategy or a mapping from states to actions that the agent follows to maximize its expected cumulative reward.

Skip Connection: Skip Connections (or Shortcut Connections) as the name suggests skips some of the layers in the neural network and feeds the output of one layer as the input to the next layers. Skip Connections were introduced to solve different problems in different architectures. In the case of ResNets, skip connections solved the degradation problem whereas, in the case of DenseNets, it ensured feature re-usability.

3 Background

Neural Architecture Search (NAS) is a subfield of machine learning (ML) and artificial intelligence (AI) that aims to automate the design of neural network architectures. The goal of NAS is to discover

optimal or near-optimal neural network architectures for specific tasks, such as image classification, object detection, language modeling, and more.

Traditionally, designing neural network architectures has been a manual and time-consuming process, requiring expertise and intuition from researchers and engineers. However, with the growing complexity and diversity of neural network architectures, manual design becomes increasingly impractical and inefficient. NAS addresses this challenge by automating the process of architecture design, thereby saving time and resources while potentially discovering architectures that outperform those designed manually.NAS has seen significant advancements in recent years, driven by both academia and industry. Techniques such as reinforcement learning-based methods (e.g., Neural Architecture Search with Reinforcement Learning - NASRL), evolutionary algorithms (e.g., Genetic Algorithm for Neural Architecture Search - GANAS), and gradient-based optimization methods (e.g., Differentiable Architecture Search - DARTS) have been proposed to automate the process of architecture design.

NAS has led to the discovery of novel architectures that achieve state-of-the-art performance across various tasks and domains. However, challenges such as high computational costs, limited generalization to unseen datasets, and the need for effective transferability of discovered architectures remain areas of active research in NAS.

4 NAS

A gradient-based method employed for discovering effective neural network architectures.Utilizes a recurrent network as the "controller" to generate the aforementioned model parameters.The neural network specified by the generated string is referred to as the "child network".The child network is trained on real data, yielding an accuracy on a validation set.The accuracy obtained serves as the reward signal for the training process.The policy gradient is computed using the reward signal to update the controller.In subsequent iterations, the controller assigns higher probabilities to architectures with higher ac curacies.Over time, the controller learns to enhance its search by favoring architectures with improved performance

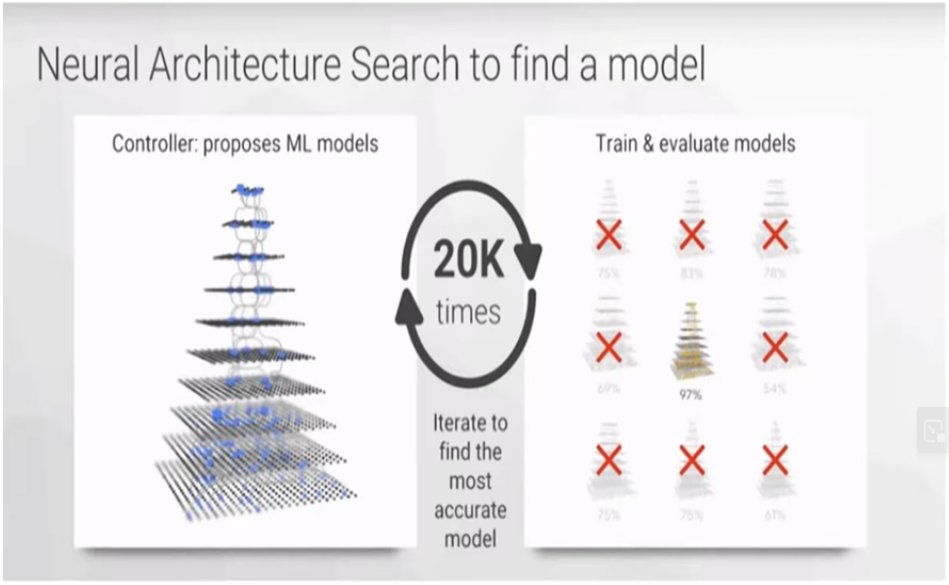


Figure 3: NAS variations

5 Importance of NAS

Neural Architecture Search (NAS) stands as a pivotal advancement in machine learning and artificial intelligence due to its capacity to automate the intricate process of designing neural network architectures. By leveraging optimization techniques and exploring vast search spaces, NAS streamlines architecture design, offering a pathway to uncover architectures that surpass manually crafted

ones. This automation not only expedites model development but also conserves significant time, computational resources, and human effort. Moreover, NAS facilitates the discovery of domain-specific architectures tailored to distinct tasks and applications, fostering innovation and pushing the boundaries of what's achievable in AI. Its scalability and adaptability across diverse hardware platforms and computational environments make it a versatile tool for both academic research and industrial applications. Additionally, NAS contributes to the development of more transferable models by uncovering architectures that capture fundamental principles of representation learning, thus enhancing generalization capabilities across tasks and datasets. Overall, NAS plays a critical role in advancing the field by revolutionizing architecture design, improving model performance, and driving innovation in machine learning and AI. Introduces a method that overcomes the limitations of existing approaches, providing greater flexibility in generating variable-length configurations for network structures.Below are the various statistics captures for NAS implementations with respect to the GPU usages, number of days the model was trained , number of parameters involved and what was error percentage captured.

Method	GPUs	Times (days)	Params (million)	Error (%)
DenseNet-BC (Huang et al., 2016)	–	–	25.6	3.46
DenseNet + Shake-Shake (Gastaldi, 2016)	–	–	26.2	2.86
DenseNet + CutOut (DeVries & Taylor, 2017)	–	–	26.2	**2.56**
Budgeted Super Nets (Veniat & Denoyer, 2017)	–	–	–	9.21
ConvFabrics (Saxena & Verbeek, 2016)	–	–	21.2	7.43
Macro NAS + Q-Learning (Baker et al., 2017a)	10	8-10	11.2	6.92
Net Transformation (Cai et al., 2018)	5	2	19.7	5.70
FractalNet (Larsson et al., 2017)	–	–	38.6	4.60
SMASH (Brock et al., 2018)	1	1.5	16.0	4.03
NAS (Zoph & Le, 2017)	800	21-28	7.1	4.47
NAS + more filters (Zoph & Le, 2017)	800	21-28	37.4	**3.65**
ENAS + macro search space	1	0.32	21.3	4.23
ENAS + macro search space + more channels	1	0.32	38.0	**3.87**
Hierarchical NAS (Liu et al., 2018)	200	1.5	61.3	3.63
Micro NAS + Q-Learning (Zhong et al., 2018)	32	3	–	3.60
Progressive NAS (Liu et al., 2017)	100	1.5	3.2	3.63
NASNet-A (Zoph et al., 2018)	450	3-4	3.3	3.41
NASNet-A + CutOut (Zoph et al., 2018)	450	3-4	3.3	**2.65**
ENAS + micro search space	1	0.45	4.6	3.54

Figure 4: NAS Statistics

6 Various Pros and Cons of NAS

Neural Architecture Search (NAS) presents several advantages in the realm of machine learning model development. One primary advantage lies in its automated approach to designing neural network architectures. By leveraging computational algorithms, NAS streamlines the process of architect neural networks, reducing the manual effort and expertise required. This automation not only accelerates the pace of model development but also allows for the exploration of a wider range of architectural configurations than traditional manual methods.

Furthermore, NAS has demonstrated its potential to yield superior performance compared to manually crafted architectures. Through systematic exploration of the architectural space, NAS algorithms can uncover novel configurations that exhibit improved accuracy, efficiency, or speed. This capability to optimize architectures for specific tasks or datasets can lead to significant advancements in model performance and ultimately enhance the quality of machine learning solutions.

Moreover, NAS offers substantial time and resource savings in model development. By automating the search process, NAS algorithms efficiently navigate the vast landscape of possible architectures, identifying promising candidates without the need for exhaustive manual experimentation. This

efficiency translates to reduced computational costs and faster iteration cycles, enabling researchers and practitioners to iterate more rapidly and explore a greater variety of architectural possibilities.

Another advantage of NAS is the transfer ability of discovered neural architectures. Architectures identified through NAS often exhibit a degree of generalized, allowing them to be adapted and applied to different tasks or domains with minimal additional effort. This transfer ability enhances the scalability and versatility of NAS-driven model development, enabling the reuse and re purposing of architectural solutions across a diverse range of machine learning applications.

While NAS offers numerous benefits, it is not without its limitations and challenges. These include computational complexity, algorithmic overhead, and the potential for over fitting or sub optimal solutions. Nonetheless, the advantages of NAS in automating and optimizing neural network design make it a valuable tool for advancing the state-of-the-art in machine learning research and applications.

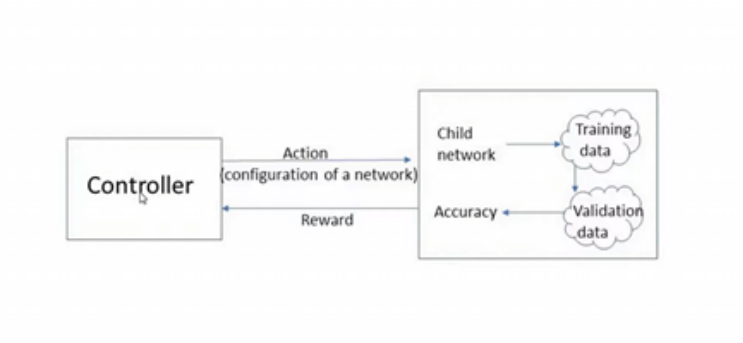


Figure 5: A basic NAS process

7 Types of NAS

Reinforcement Learning (RL): Reinforcement Learning-based NAS methods treat the architecture search process as a sequential decision-making problem. In RL-based NAS, an agent interacts with an environment, which represents the search space of neural architectures, and learns to select architectures that maximize a reward signal indicative of performance on a given task. The agent's policy is updated through trial-and-error exploration, guided by reinforcement learning algorithms such as Q-learning, Policy Gradient methods, or Actor-Critic models. RL-based NAS has the advantage of adaptability to various search spaces and tasks, as well as the potential to discover novel architectures that surpass manually designed ones. However, RL-based approaches often require significant computational resources and careful tuning of hyperparameters to achieve good performance.

Differentiable Architecture Search (DARTS): Differentiable Architecture Search (DARTS) is a gradient-based approach to NAS that enables end-to-end training of both the neural architecture and its associated weights. In DARTS, the architecture search space is defined as a continuous relaxation of discrete decisions, allowing for efficient gradient-based optimization using standard backpropagation. DARTS employs a differentiable operation to select and combine architectural components, enabling the search process to be fully differentiable. This allows for the efficient exploration of the architecture space using gradient descent methods. DARTS has gained popularity due to its simplicity, efficiency, and effectiveness in discovering high-performing architectures. However, it may suffer from issues such as instability during training and limitations in capturing the full complexity of the architecture space.

Random Search: Random search involves randomly sampling neural architectures from a predefined search space and evaluating their performance on a validation dataset. Despite its simplicity, random search can be effective in finding good architectures when coupled with a sufficiently large search space and computational resources.

Evolutionary Algorithms: Evolutionary algorithms, such as genetic algorithms or evolutionary strategies, mimic the process of natural selection to evolve populations of neural architectures

over successive generations. These algorithms typically involve selection, mutation, and crossover operations to iteratively improve architectures based on their fitness.

Gradient-based Optimization: Gradient-based optimization methods optimize neural architectures by directly differentiating through the architecture space and updating architectural parameters using gradient descent. Techniques like reinforcement learning, policy gradients, and gradient-based optimization have been employed to train neural architecture search agents.

Bayesian Optimization: Bayesian optimization techniques leverage probabilistic models to efficiently search for optimal neural architectures. These methods maintain a probabilistic model of the objective function and iteratively select new candidate architectures to evaluate based on the model's predictions.

Ensemble-based NAS: Ensemble-based NAS methods combine multiple neural architectures to form an ensemble, leveraging the diversity of individual architectures to improve overall performance. These methods often involve training and combining multiple architectures to exploit their complementary strengths and mitigate weaknesses.

Progressive NAS: Progressive NAS starts with a small, simple neural network architecture and progressively grows and refines the architecture based on task performance. This incremental approach allows for the exploration of increasingly complex architectures while adapting to the requirements of the task at hand.

Network Morphism: Network morphism involves dynamically modifying neural network architectures during training to adapt to changes in the input data or task requirements. This approach allows networks to evolve and specialize over time, improving their adaptability and performance in varying conditions.

Hierarchical NAS: Hierarchical NAS methods organize the search space of neural architectures into hierarchical structures, enabling more efficient exploration and optimization. By decomposing the search process into hierarchical levels, these methods can reduce the computational complexity of NAS while maintaining the ability to discover complex architecture.

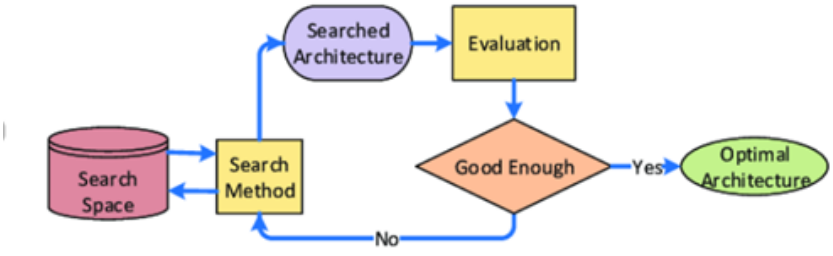
Neural Evolutionary Strategies: Neural evolutionary strategies combine neural networks with evolutionary algorithms to search for optimal neural architectures. Instead of directly optimizing architectural parameters, these methods evolve populations of neural networks, which are evaluated based on their performance on a given task.

Meta-Learning and Transfer Learning: Meta-learning approaches aim to learn a learning algorithm that can adapt to new tasks or datasets with minimal additional training. In the context of NAS, meta-learning techniques can be used to learn a neural architecture search algorithm that generalizes across different tasks or domains.

One-Shot and Progressive Methods: One-shot methods aim to train a single neural network that can perform architecture search and task learning simultaneously. Progressive methods start with a small, simple network and progressively grow and refine the architecture based on task performance.

8 NAS Processes

The objective is to use a controller(RNN) to generate architectural hyper parameters for neural networks.The controller is implemented as a recurrent neural network (RNN) for flexibility in generating hyper parameters.The flexibility of the controller allows it to adapt to various architectural configurations.The controller generates hyper parameters in the form of a sequence of tokens.Each token in the sequence corresponds to a specific architectural hyper parameter, enabling a structured representation.The approach is adaptable to different network structures, as indicated by the focus on predicting feed forward neural networks with convolutions layers.The described controller can be applied to a range of tasks involving the generation of architectural hyper parameters for neural networks.



Basic building blocks of Neural Architecture Search methods.

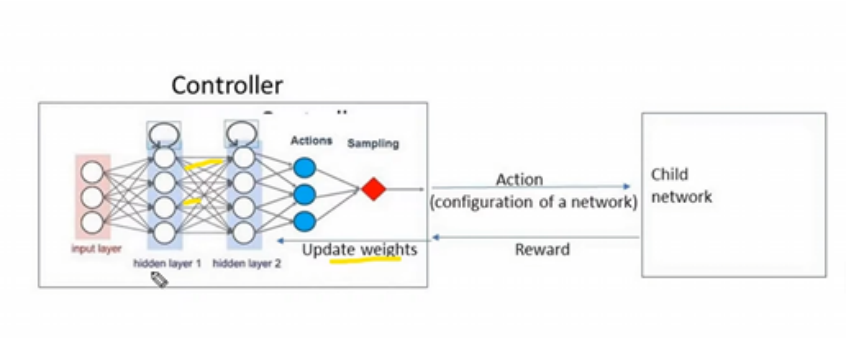


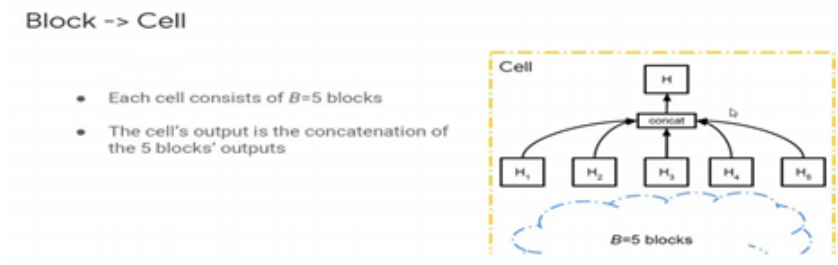
Figure 6: A picture depicting in depth details of NAS process

9 NAS Methods

The recurrent network can be trained with a policy gradient method to maximize the expected accuracy of the sampled architectures.

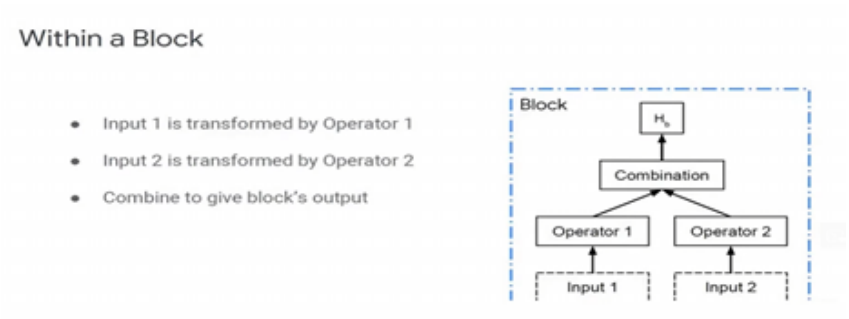
Figure 7: Representation of block to cell to network

Forming skip connections to increase model complexity and using a parameter server approach to speed up training.



A controller to generate architectural hyper-parameters of neural networks. The controller is implemented as RNN.To predict feed forward neural networks with only convolutional layers, we can use

the controller to generate their hyper parameters as a sequence of tokens.The controller RNN finishes generating an architecture, a neural network with this architecture is built and trained.



The parameters of the controller RNN, theta c, are then optimized in order to maximize the expected validation accuracy of the proposed architectures.

10 NAS Algorithm

In this document we are going to talk about mainly reinforcement learning algorithm and dart based NAS algorithms.

RL Based

Algorithm 1 Reinforcement Learning-based NAS

Initialize agent policy π
Initialize agent parameters θ
Define environment E
Define reward function R
Define hyperparameters: ϵ, α, γ
for episode in range(num_episodes) **do**
    Reset environment to initial state: $state = E.reset()$
    Initialize episode-specific variables: $episode\_reward = 0$
    while not done **do**
        if random_uniform() < ϵ **then**
            Exploration: Randomly sample action: $action = E.sample\_random\_action()$
        else
            Exploitation: Select action using agent's policy: $action = \pi(state, \theta)$
        end if
        Take action in environment and observe next state and reward: $next\_state, reward, done = E.step(action)$
        Update episode reward: $episode\_reward += reward$
        Update agent's policy using RL algorithm:
            $td\_target = reward + \gamma \times \max(\pi(next\_state, \theta))$ if not done else reward
            $td\_error = td\_target - \pi(state, \theta)[action]$
            $\theta[action] += \alpha \times td\_error$
        Transition to next state: $state = next\_state$
    end while
    Update exploration rate (optional, if using epsilon-greedy policy): $\epsilon * = decay\_rate$
    Print episode results: "Episode:", episode, "Reward:", episode_reward
end for

DART Based

Algorithm 2 Differentiable Architecture Search (DARTS)

Initialize the architecture weights α and network parameters θ
Initialize the loss function L
Define the search space $\mathcal{A}$ of possible operations
Define the optimization algorithm and hyper parameters (e.g., SGD, learning rate η)
for each training iteration **do**
    Sample a mini-batch of data (x, y) from the training set
    Forward pass through the network with architecture weights: $y_{\text{model}} = f(x; \alpha, \theta)$
    Compute the loss: $L_{\text{model}} = L(y_{\text{model}}, y)$
    Compute the gradients of the loss with respect to the network parameters: $\nabla_\theta L_{\text{model}}$
    Update the network parameters using gradient descent: $\theta \leftarrow \theta - \eta \times \nabla_\theta L_{\text{model}}$
    for each edge in the architecture **do**
        Compute the gradients of the architecture weights using the validation set:
            $\nabla_\alpha L_{\text{val}} = \nabla_\alpha L(y_{\text{val}}, f_{\text{val}}(x_{\text{val}}; \alpha, \theta))$
    end for
    Update the architecture weights using gradient descent: $\alpha \leftarrow \alpha - \eta \times \nabla_\alpha L_{\text{val}}$
end for
Return the final optimized architecture weights α

11 Training with RL

The REINFORCE rule, also known as the policy gradient theorem, is a fundamental concept in reinforcement learning. It is used to update the parameters of a policy in order to maximize expected rewards.

Let θ represent the parameters of the policy function $\pi_\theta(a|s)$, where $\pi_\theta(a|s)$ denotes the probability of taking action a given state s under the policy parameterized by θ.

The objective is to maximize the expected return $J(\theta)$:

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[R(\tau)],$$

where $\tau = (s_0, a_0, r_1, \ldots)$ is a trajectory, and $p_\theta(\tau)$ is the probability of trajectory τ under policy π_θ.

The REINFORCE rule updates the policy parameters θ using the gradient of the expected return with respect to θ:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) \cdot R(\tau)].$$

The Williams formula simplifies this expression to:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot \hat{Q}(s_t, a_t)],$$

where $\hat{Q}(s_t, a_t)$ is an estimate of the action-value function at state s_t and action a_t. This estimate can be obtained from the rewards collected during trajectory τ.

The REINFORCE rule with the Williams formula is used in policy gradient methods, such as vanilla policy gradients and actor-critic methods, to update the policy parameters in the direction that increases the expected return.The overall expression represents the policy gradient objective, where the goal is to adjust the parameters C of the policy to maximize the expected cumulative reward(accuracy) , taking into account multiple trajectories and a baseline to reduce variance in the gradient estimate. This type of formulation is commonly used in reinforcement learning for training policies in sequential decision-making tasks

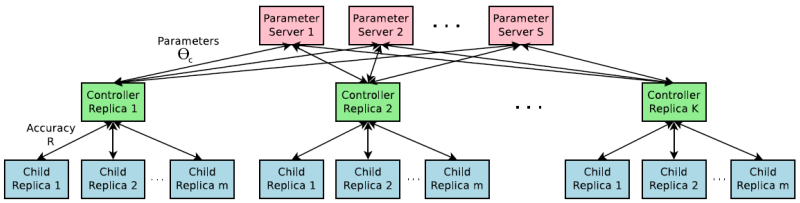


Figure 8: Distributed controller and child network training method

12 Accelerate Training with Parallelism and Asynchronous Updates

Accelerating training with parallelism and asynchronous updates is a strategy employed in deep learning to expedite model training, particularly for large models and datasets. Parallelism involves distributing computations across multiple processing units, such as CPUs or GPUs, allowing tasks to be performed simultaneously. This is advantageous for handling the computational demands of training deep neural networks.

One facet of parallelism is data parallelism, where multiple workers process different mini-batches of data concurrently. Each worker computes gradients independently and periodically synchronizes model parameters with a central server. By dividing the data across workers and leveraging parallel computation, data parallelism enables more efficient utilization of computational resources and can significantly expedite training, especially for large datasets.

Another aspect is model parallelism, which involves partitioning the model and assigning different parts to various processing units. This approach is beneficial for models with architectures too large to fit entirely into the memory of a single device. By distributing computations across multiple devices, model parallelism facilitates training of larger models that might otherwise be impractical to train on a single device.

In conjunction with parallelism, asynchronous updates further enhance training speed. In asynchronous updates, workers independently compute gradients and update model parameters without waiting for synchronization. This contrasts with synchronous training, where workers must wait for gradients to be averaged before updating parameters. Asynchronous updates reduce communication overhead and can lead to faster convergence, especially in scenarios with heterogeneous computing resources or where communication bandwidth is limited.

In summary, leveraging parallelism and asynchronous updates in deep learning allows for more efficient training of large models on large datasets. These techniques harness the capabilities of modern computational architectures to expedite the optimization process and enable the training of complex models within reasonable time frames.

13 Increasing architecture complexity with skip connections and other layer

Increasing architecture complexity often involves incorporating advanced architectural elements like skip connections and additional layers in neural networks. Skip connections, also known as residual connections, are a key architectural innovation that facilitate the training of very deep neural networks. These connections bypass one or more layers by adding the input to the output of the subsequent layers. This allows for better flow of gradients during back propagation, mitigating the vanishing gradient problem and enabling the training of deeper networks. As a result, skip connections promote the construction of deeper architectures without sacrificing optimization performance, leading to improved model capacity and generalization ability.

Moreover, introducing additional layers beyond traditional architectures can further enhance model expressiveness and capture complex patterns in data. For instance, convolutional neural networks (CNNs) often incorporate multiple layers such as convolutional, pooling, and fully connected layers to extract hierarchical features from images. Similarly, recurrent neural networks (RNNs) may include multiple layers of recurrent units to capture temporal dependencies in sequential data. By increasing the number of layers, the network gains the capacity to learn increasingly abstract and sophisticated

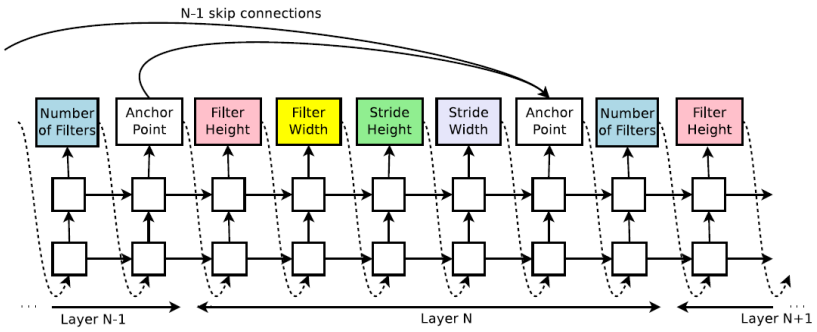


Figure 9: Demonstration of skip-connection

NAS Performance

NAS	AutoKeras (%)	ENAS (macro) (%)	ENAS (micro) (%)	DARTS (%)	NAO-WS (%)
Fashion-MNIST	91.84	95.44	95.53	**95.74**	95.20
CIFAR-10	75.78	95.68	**96.16**	94.23	95.64
CIFAR-100	43.61	78.13	78.84	**79.74**	75.75
OUI-Adience-Age	63.20	**80.34**	78.55	76.83	72.96
ImageNet-10-1	61.80	77.07	79.80	**80.48**	77.20
ImageNet-10-2	37.20	58.13	56.47	60.53	**61.20**

Figure 10: NAS Performance for various datasets and methods

representations of the input data, which can be crucial for tasks requiring nuanced understanding or high-dimensional feature extraction.

However, while increasing architecture complexity can lead to improved performance and representation power, it also comes with challenges. Deeper architectures with skip connections and additional layers introduce more parameters, which can increase computational and memory requirements during training and inference. Moreover, deeper networks are prone to over fitting, especially when the training data is limited or noisy. Consequently, careful regularization techniques such as dropout, batch normalization, and weight decay may be necessary to prevent over fitting and ensure effective training.

In summary, incorporating skip connections and additional layers into neural network architectures allows for the construction of deeper and more expressive models capable of capturing intricate patterns in data. While these architectural enhancements can lead to improved performance, they also require careful consideration of computational resources and regularization techniques to mitigate potential challenges such as over fitting.

14 NAS Performance Analytics

The below table represents statistics captured for various variants of the NAS.The ENAS on CIFAR-10 is close to 96.16.Similarly DART performance on Fashion-MNIST is 95.75

15 Analysis

Drawbacks of the Paper:

High Computational Cost: The RL-based NAS approach proposed in the paper requires a significant amount of computational resources. Training thousands of neural architectures to convergence is computationally expensive, limiting its accessibility to researchers with constrained resources.

Limited Exploration: The RL agent's exploration strategy might be limited, leading to sub optimal architectures or premature convergence to local optima. This limitation could result in missed opportunities for discovering more efficient or effective architectures.

Evaluation Metrics: The paper primarily evaluates the discovered architectures on benchmark datasets like CIFAR-10 and Penn Treebank. While these datasets are standard in the field, they may not fully capture a model's performance across diverse real-world scenarios.

Transferability: The effectiveness of the discovered architectures across different tasks or domains is not extensively explored. The paper could benefit from evaluating the transferability of the architectures to various tasks beyond the ones considered.

Importance of the Method:

Automated Architecture Discovery: The RL-based NAS approach automates the process of architecture design, reducing the need for manual intervention and bias in architectural choices. This automation can accelerate progress in machine learning research by efficiently exploring a vast space of possible architectures.

Efficiency Improvement: By optimizing architectures for specific tasks, the proposed method aims to improve efficiency in terms of both computational resources and model performance. This is crucial for deploying models in resource-constrained environments.

Generalization Potential: Despite limitations, the RL-based NAS method holds potential for discovering architectures that generalize well across different tasks and datasets, paving the way for more robust and versatile machine learning models.

Future Applications:

Customized Model Architectures: The automated architecture search facilitated by RL-based NAS could lead to the development of tailored models for specific applications, such as computer vision, natural language processing, and speech recognition.

Resource-Constrained Environments: The efficient architectures discovered through NAS could be deployed in resource-constrained environments, including mobile devices, edge devices, and IoT devices, enabling on-device inference with minimal computational overhead.

Continued Research in NAS: The paper's contributions highlight the potential of RL-based NAS as a promising avenue for future research in automated machine learning (AutoML), with applications ranging from model compression to domain-specific optimization.

Room for Improvement:

Sample Efficiency: Improving the sample efficiency of RL-based NAS methods could reduce the computational cost and accelerate the architecture search process. Techniques such as better exploration strategies or surrogate models could enhance sample efficiency.

Robustness and Generalization: Future research should focus on ensuring that the discovered architectures generalize well across diverse datasets and tasks. Incorporating techniques for encouraging exploration and diversity in architecture search could improve robustness and generalization.

Real-World Evaluation: The effectiveness of NAS methods should be evaluated in real-world scenarios with practical constraints and considerations, such as deployment environments, data distribution shifts, and adversarial robustness.

16 Conclusion

In summary, while the RL-based NAS approach presented by Zoph and Le offers significant advancements in automated architecture search, there are several challenges and areas for improvement that need to be addressed to realize its full potential in practical applications. Continued research and innovation in NAS methodologies are essential for overcoming these challenges and making automated architecture search more accessible, efficient, and effective.

Task Distribution

Ashish Verma: Solely worked on power point presentation as well as building the latex document.

References

[1] Neural architecture and search methods. https://www.geeksforgeeks.org/neural-architecture-and-search-methods/.

[2] Author. Neural architecture search and optimization. https://www.youtube.com/watch?v=NQj5TkqX48Q.

[3] Author. Introduction to neural architecture search. https://www.youtube.com/watch?v=UlvkBZdOhpg.

[4] Nas overview. https://www.automl.org/nas-overview/.

[5] Neural architecture search. https://en.wikipedia.org/wiki/Neural_architecture_search.

[6] What is neural architecture search? https://www.infoworld.com/article/3648408/what-is-neural-architecture-search.html.

[7] Neural architecture search: Evolution, methods, and tools. https://deci.ai/neural-architecture-search/.

[8] What is neural architecture search? https://www.oreilly.com/content/what-is-neural-architecture-search/.

[9] Basic building blocks of neural architecture search methods. https://www.researchgate.net/figure/Basic-building-blocks-of-Neural-Architecture-Search-methods_fig1_352621860.

[10] Comparison of nas methods. https://nni.readthedocs.io/en/stable/sharings/nas_comparison.html.

[11] Author. Cheat sheet: Recurrent neural networks. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

[12] Reinforcement learning 101. https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292.