Dynamic Architectural Optimization of Artificial Neural Networks

Kaitlin Maile

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Université Toulouse 1 Capitole (UT1 Capitole)*

Présentée et soutenue le *04/10/2023* **par :**

Kaitlin MAILE

Dynamic Architectural Optimization of Artificial Neural Networks

JURY

Carola DOERR	Rapportrice
Decebal C. MOCANU	Rapporteur
Marc SCHOENAUER	Examinateur
Rufin VAN RULLEN	Examinateur
Frank HUTTER	Examinateur
Hervé LUGA	Directeur de Thèse
Dennis G. WILSON	Co-encadrant de Thèse

École doctorale et spécialité :
MITT : Domaine STIC : Intelligence Artificielle
Unité de Recherche :
IRIT : Institut de Recherche en Informatique de Toulouse
Directeur(s) de Thèse :
Hervé LUGA et *Dennis G. WILSON*
Rapporteurs :
Carola DOERR et *Decebal C. MOCANU*

Abstract

Despite the significant and exponential advancements in artificial neural networks (ANNs), their current capabilities still fall short of human-like intelligence. Many assumptions of current artificial learning approaches diverge from the observed characteristics of the brain, such as conventionally static and hand-designed ANN architectures opposed to dynamic and self-modifying biological connectivity. This thesis aims to further progress by exploring structural learning as a brain-inspired tool for augmenting the power of ANNs, taking initial steps of expanding architectural search spaces and learning paradigms to real practical applications. Towards this end, the proposed structural learning framework unifies multiple subfields of artificial intelligence research and identifies key foundational challenges that are investigated in the subsequent work. Towards architectural search space generalization, equivariance-aware neural architecture search (NAS) optimizes the architectural constraints imposed by partial equivariance to symmetry groups, enhancing the performance and generalization of ANNs for tasks exhibiting symmetries. Algorithmic improvements to differentiable NAS, focusing on dynamic scheduling and regularization, enhance the efficiency and reliability of the search process. Neurogenesis in ANNs, a relatively unstudied problem, is decomposed into scheduling and initialization decisions towards a suite of neurogenesis strategies that enable the dynamic growth of performant networks. Finally, aspects of all of the preceding findings are synthesized towards architectural optimization in dynamic learning environments such as transfer learning. The findings and methodologies presented have the potential to significantly impact the standard pipeline of ANNs, reducing engineering, training, and deployment costs while increasing their efficacy and power for practitioners.

Résumé

Malgré les progrès considérables et exponentiels des réseaux neuronaux artificiels (ANNs), leurs capacités actuelles sont encore loin de l'intelligence humaine. De nombreuses hypothèses des approches actuelles de l'apprentissage artificiel divergent des caractéristiques observées du cerveau, telles que les architectures ANN conventionnellement statiques et conçues à la main, opposées à la connectivité biologique dynamique et auto-modifiante. Cette thèse vise à faire avancer les choses en explorant l'apprentissage structurel en tant qu'outil inspiré du cerveau pour augmenter la puissance des ANNs, en prenant des mesures initiales pour étendre les espaces de recherche architecturaux et les paradigmes d'apprentissage à des applications pratiques réelles. À cette fin, le cadre d'apprentissage structurel proposé unifie plusieurs sous-domaines de la recherche en intelligence artificielle et identifie les principaux défis fondamentaux qui sont étudiés dans les travaux ultérieurs. En ce qui concerne la généralisation de l'espace de recherche architecturale, la recherche d'architecture neuronale (NAS) tenant compte de l'équivariance optimise les contraintes architecturales imposées par l'équivariance partielle aux groupes de symétrie, améliorant ainsi les performances et la généralisation des ANNs pour les tâches présentant des symétries. Les améliorations algorithmiques apportées à la recherche d'architecture neuronale différentiable, axées sur la planification dynamique et la régularisation, renforcent l'efficacité et la fiabilité du processus de recherche. La neurogenèse dans les ANNs, un problème peu étudié, est décomposée en décisions d'ordonnancement et d'initialisation vers une suite de stratégies de neurogenèse qui permettent la construction automatique et dynamique de réseaux performants. Enfin, les aspects de tous les résultats précédents sont synthétisés en vue d'une optimisation architecturale dans des environnements d'apprentissage dynamiques tels que l'apprentissage par transfert. Les résultats et les méthodologies présentés ont le potentiel d'avoir un impact significatif sur le pipeline standard des RNA, en réduisant les coûts d'ingénierie, de formation et de déploiement tout en augmentant leur efficacité et leur puissance pour les praticiens.

Acknowledgements

I would like to thank my advisors, Dennis and Hervé. Their help was imperative from our humble beginnings as a remote internship to test the waters before committing to the full PhD, then continuously navigating french bureaucracy to successfully relocate to France and secure funding for the thesis and associated travels. Their advisement style adapted to my growth as a researcher, dynamically providing just the right amount of constructive guidance.

I would like to thank my parents, Keith and Sandi. Their persistent support and inspiration has made my entire academic path possible, evidenced by their willingness to chauffeur me to a generous number of academic and extracurricular activities throughout my childhood as well as their unfaltering support at each life path decision.

I would like to thank the rest of my jury members: Carola, Decebal, Marc, Rufin, and Frank, for their role in the final stages of the doctorate and all of the intriguing scientific discussions with each one leading up to the defense.

I would like to thank the SuReLI lab at ISAE-SUPAERO and the AMLab at the University of Amsterdam for providing research environments that both catalyzed insightful discussions as well as offered a cordial social opportunities to relax in-between.

I am grateful for all of my research experiences leading up to my doctoral studies. David helped me navigate the world of research and then of graduate school applications to transition from biomedical engineering to computer science. Risto and Manish then supported my growth as a graduate researcher at UT Austin as well as my decision to begin this thesis.

I would like to thank the remaining co-authors, collaborators, and connections made at conferences, internships, and beyond that provided inspirational discussions and useful advice.

I would like to thank all of the old and new friends I have made along the way, in addition to connections via sports such as climbing partners and ultimate frisbee teammates, for keeping me sane and providing so many unforgettable experiences.

Contents

1 Introduction

Artificial intelligence algorithms, and specifically artificial neural networks (ANNs), have been able to grow exponentially in power and complexity since their roots almost a century ago [1]. This is due to innovative automations such as backpropagation that replace some of what has been previously hand designed, as well as breakthroughs in computational power [2]. However, the currently realizable potential of ANNs in human-like intelligence is still nowhere near that of the brain, their original inspiration. Additionally, many assumptions of current learning approaches such as structurally static learning and isolated training from scratch have diverged from what is observed in the brain. This history has produced ANNs with a notably different intelligence from humans, excelling at computational and quantitative tasks yet failing at analytical and abstract aspects. While such assumptions have thus far helped push the frontier of artificial intelligence to its current state by adapting to the characteristics of artificial hardware and software, they may become a bottleneck preventing more powerful and adaptive ANNs that don't require hand-designed architectures.

The architecture of an ANN determines what operations are performed on the input data and how they are ordered to compute the output. An ANN structured specifically for a given domain can approximate a function more practically and efficiently in storage and training time than a generic wide shallow network [3, 4], even though the latter are theoretically capable of approximating any function [5, 6]. Even dense ANNs, or multi-layer perceptrons (MLPs), use structural priors and are not completely dense, as each neuron is only connected to the neurons in the immediately preceding and following layers, rather than all neurons in a network. This imposes a basal amount of structure on the network. Convolutional networks are even sparser than this, limiting neurons to have only spatially local connections and imposing weight-sharing on these connections to create translational equivariance for each layer. Skip-connections are a further structure that can be represented within the broader representation of layers connected in series. Although each of these structures is simply a special case of more generic functions, their constraints yield valuable benefits during the learning process.

The brain also shows structural specialization with hierarchical organization in task-specific regions, such as visual cortex [7], suggesting the benefit of structure for guiding learning. Convolutional layers were originally inspired by the visual cortex [8, 9]. The functional form conventionally used in modern ANNs is provably part of the most general and expressive class of linear functions with translational equivariance in the image domain [10], only adding a constraint of kernel size to enforce locality. This concurrence supports the utility of looking to the brain for inspiration, even though the clear divergence of ANNs from their biological roots must be appropriately considered.

Deeps ANNs are usually hand-designed, structurally static, and used generally across many related tasks within a domain. Even with specialized structures, they are known to be over-parameterized [11, 12], using more parameters than necessary for their performance on a specific task after training. Over-parameterization is conjectured to help gradient descent of parameters of an ANN converge to the global optima for a given training dataset, task, and objective function [13], particularly given that the architectures are reused for many tasks due to prohibitive engineering costs for further specialization. However, this over-parameterization also comes at a cost of space and time efficiency, during training and deployment. Using even more specialized architectures, beyond what is feasible through manual design, that themselves support parameter optimization could evade this cost and increase the efficacy of the ANN in a specific task.

The brain structures itself dynamically using both genetic and environmental information. Our DNA encodes processes that support the formation of the visual cortex, but only in the presence of typical visual input during development [14]. Otherwise, the brain learns to repurpose the unused regions for other structures and functions [15]. This emphasizes the important interplay of the learning algorithm

in addition to the data towards adaptability, such that specialized function-permissive structure may be learned dynamically.

Automated and dynamic specialized design of ANN architectures is achievable through *structural learning*: a type of architectural optimization where both the architecture and the parameters of that architecture are cooperatively learned in a single process. This manuscript investigates structural learning as a brain-inspired tool for enhancing the learning process and augmenting the power and adaptability of ANNs.

1.1 Current Challenges and Goals

The overarching challenges presented subsequently have been identified in the current state of literature and will be detailed throughout this manuscript, reflected in the accomplished and future work.

1.1.1 Improving the Automation of Neural Networks

Furthering the automation of ANNs beyond just weight optimization would expand their applicability, adaptability, and potential. Structural learning is one of the more brain-inspired approaches, compared to others such as hyperparameter optimization, for tackling this challenge. As computational capacities continue to improve, research into how to implement structural learning in a manner suited to artificial optimization could maximize the utilization of ANNs without the bottleneck of manual architecture design for further innovation in artificial intelligence. Even within existing structural learning approaches, the schedule of applying structural changes must be hand-designed, prompting the desire for automated scheduling: what structural changes to perform and when, where, how, and why to perform them.

1.1.2 Adaptive Learning

Architectural optimization permits dynamic architectural adaptations over learning, adjusting not only to the specific learning environment itself but also how it may be changing. Structural learning for ANNs is often studied *in silica*, using many assumptions to isolate and simplify this difficult process. The brain performs structural learning *in vivo*, notably with self-learned weak supervision and internal rewards. Exploring structural learning in more complex learning environments with fewer simplifying assumptions, such as continual learning and reinforcement learning, can support more adaptable models towards more complex and realistic applications. Automated scheduling is also crucial in this challenge, as the structural learning dynamics should adapt to the those of both the learning environment and the model itself during training.

1.1.3 Expanding the Domains and Search Spaces of Structural Learning

Structural learning has thus far mostly been studied within the same image classification benchmark tasks and structural search spaces akin to existing convolutional architectures. While this allows a deeper understanding, it is not permissive towards applying it to the vast breadth of potential domains beyond simple image classification. Enabling application in more powerful search spaces, such as search through equivariant operations beyond translation to further capitalize on symmetries and structure present in the domain, could further both the study of structural learning as well as of the respective domains of interest.

1.2 Contributions

This manuscript combines the work of my doctorate into a cohesive thread and presents my future plans. My contributions to the field of deep learning are presented in the following sections.

1.2.1 Sections 2.2 and 3 - Structural Learning in Artificial Neural Networks: A Neural Operator Perspective [16]

Over the history of ANNs, only a minority of algorithms integrate structural changes of the network architecture into the learning process. Modern neuroscience has demonstrated that structural change is an important part of biological learning, with mechanisms such as synaptogenesis and neurogenesis present even in adult brains and considered important for learning. Despite this history of artificial methods and biological inspiration, and furthermore the recent resurgence of neural methods in deep learning, there are relatively few current ANN methods which include structural changes in learning compared to those that only adjust synaptic weights during the training process. We aim to draw connections between different approaches of structural learning that have similar abstractions in order to encourage collaboration and development. In this review, we provide a survey on structural learning methods in deep ANNs, including a new neural operator framework from a cellular neuroscience context and perspective aimed at motivating research on this challenging topic. We first give a background on biological developmental processes in the brain. We then provide an overview of ANN methods which include structural changes within the neural operator framework in the learning process, characterizing each neural operator in detail and drawing connections to their biological counterparts. Finally, we present overarching trends in how these operators are implemented and discuss the open foundational challenges in structural learning in ANNs towards the overarching problems presented in Section 1.1 and their respective solutions presented in this manuscript.

1.2.2 Section 4 - Architectural Optimization over Subgroups for Equivariant Neural Networks [17]

Incorporating equivariance to symmetry groups as a constraint during neural network training can improve performance and generalization for tasks exhibiting those symmetries, but such symmetries are often not perfectly nor explicitly present. This motivates algorithmically optimizing the architectural constraints imposed by equivariance. We propose the equivariance relaxation morphism, which preserves functionality while reparameterizing a group equivariant layer to operate with equivariance constraints on a subgroup, as well as the $[G]$-mixed equivariant layer, which mixes layers constrained to different groups to enable within-layer equivariance optimization. We further present evolutionary and differentiable neural architecture search (NAS) algorithms that utilize these mechanisms respectively for equivariance-aware architectural optimization. Experiments across a variety of datasets show the benefit of dynamically constrained equivariance to find effective architectures with approximate equivariance. This work directly tackles the challenge of expanding the domain and search space of structural learning, utilizing optimizable partial equivariance as structural bias.

1.2.3 Section 5 - DARTS-PRIME: Regularization and Scheduling Improve Constrained Optimization in Differentiable NAS [18]

Differentiable Architecture Search (DARTS) is a recent neural architecture search (NAS) method based on a differentiable relaxation of the discrete architectural search space [19]. Due to its success, numerous variants analyzing and improving parts of the DARTS framework have recently been proposed. By considering the problem as a constrained bilevel optimization, we present and analyze DARTS-PRIME, a variant including improvements to architectural weight update scheduling and regularization towards discretization. We propose a dynamic schedule based on per-minibatch network information to make architecture updates more informed, as well as proximity regularization to promote well-separated discretization. Our results in multiple domains show that DARTS-PRIME improves both performance and reliability, comparable to state-of-the-art in differentiable NAS. These proposed contributions work together towards the challenges of automation and adaptive learning, investigating informed scheduling for structural learning.

1.2.4 Section 6 - When, Where, and How to Add New Neurons to ANNs [20]

Neurogenesis in ANNs is an understudied and difficult problem, even compared to other forms of structural learning like pruning. By decomposing it into triggers and initializations, we introduce a

framework for studying the various facets of neurogenesis: when, where, and how to add neurons during the learning process. We present the Neural Orthogonality (NORTH*) suite of neurogenesis strategies, combining layer-wise triggers and initializations based on the orthogonality of activations or weights to dynamically grow performant networks that converge to an efficient size. We evaluate our contributions against other recent neurogenesis works across a variety of supervised learning tasks. This work expands the search space of structural learning by furthering our understanding of neural creation and empowering its use in architectural optimization algorithms, as explored in the following work.

1.2.5 Section 7 - Neural Growth and Pruning in Dynamic Learning Environments [21]

Training ANNs under the shifting distributions of dynamic learning environments can be augmented by dynamic architectural adjustments in addition to the standard parameter tuning. Towards effective yet efficient models, we study neural grow-and-prune in the basic shifting distribution case of transfer learning: adapting a generically pre-trained model to a target dataset. We propose the Dynamic Neural Optimization (DYNO) grow-and-prune algorithm, which dynamically performs neural pruning to remove neurons that are dormant or redundant and neural growth to add orthogonal features during fine-tuning. Our experiments across a variety of transfer tasks show that DYNO yields an efficient yet performant network with dynamically shaped layers. This work further tackles the challenge towards dynamic structural learning, adapting models during the distribution shift of changing tasks.

1.3 Manuscript Outline

This manuscript begins by summarizing the overall challenges identified in the structural learning literature in Section 1.1 and presenting the contributions towards those goals in Section 1.2. Next, a background on both artificial and biological neural networks, particularly from a structural learning perspective, is given in Section 2. A review on structural learning is presented in Section 3 to set the stage for the following experimental works. Section 4 proposes structural tools that permit architectural optimization of partially equivariant neural networks, while Section 5 proposes algorithmic improvements to differentiable neural architecture search, towards dynamic scheduling. A foundational study on when, where, and how to perform neurogenesis is presented in Section 6, followed by an extension towards structural grow-and-prune algorithms for dynamic learning environments in Section 7. Finally, the overall conclusions and future work to follow these projects are presented in Section 8.

2 A Background on the Current State of Neural Networks

The following section serves as a brief background for artificial and biological networks, particularly in the context of structural learning.

2.1 Artificial Neural Networks

Artificial neural networks (ANNs) are a family of machine learning methods inspired by the biological brain. Their origins trace back to computational models of neurons [1], inspiring the first artificial neural networks known as perceptrons [22]: simple, hand-designed diagrams of nodes and weighted connections. Over decades, these evolved into the multi-layer perceptrons, a simple type of ANN, still used today [23].

2.1.1 Basic Functionality

The basic unit of a neural network is a neuron, which is a simplified abstraction of it's biological counterpart. A neuron is characterized by the incoming connections with synaptic weights and bias weight $\mathbf{w}$, activation function f, and outgoing connections. The output of this neuron will be the synaptically weighted sum of the inputs $\mathbf{x}$ and bias b, all passed through the activation function, portrayed in Figure 1a. For a given input, this value passed onto outgoing connections from a neuron is called its activation.

In simple fully-connected feed-forward ANNs, neurons are organized in layers, taking input from all neurons from the previous layer and outputting to all neurons in the subsequent layer, shown in Figure 1b. The inputs to the ANN are treated like a layer, such that the subsequent layer takes their value as inputs with their corresponding weights. Likewise, the output neurons of the ANN receive weighted connections from the previous layer and compute their activation as the output of the ANN.

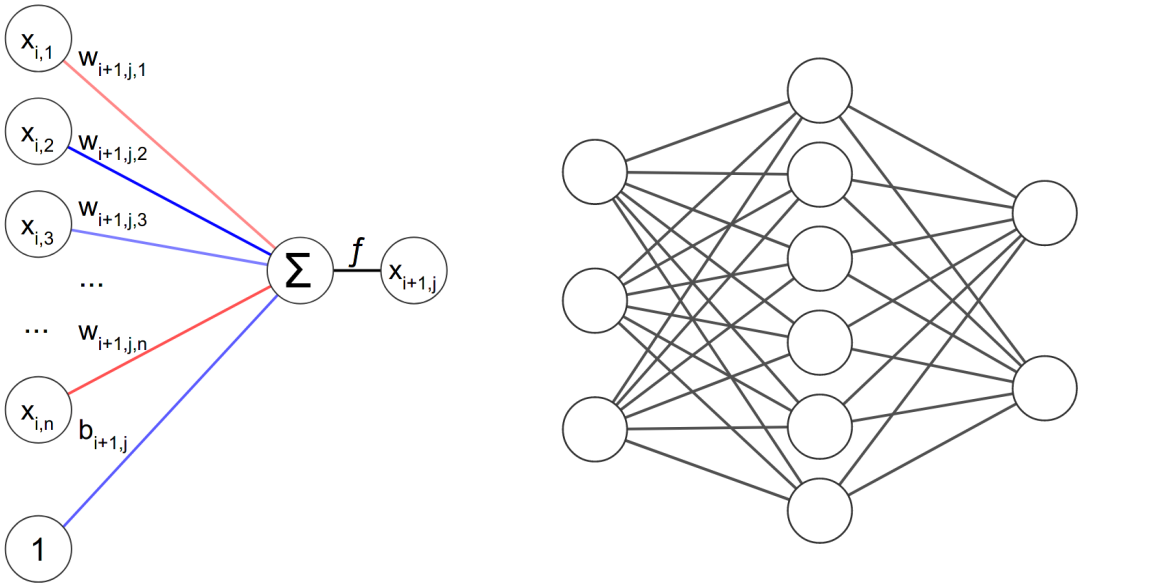
The activation function serves multiple purposes. In hidden layers, or the layers not including the input and output, it creates nonlinearity, which increases the expressive power added by each layer. Without nonlinearity, the neural network would only be capable of linear functions with no added benefit from depth. Common choices include the Rectified Linear Unit (ReLU) [24] and Sigmoid activation functions. In the output layer, the activation function choice depends on the task. For classification networks, a Softmax function is often used to convert the output activations, with dimensionality equal to the number of classes, to probabilities.

The ANNs discussed thus far are time-invariant, one of the key differences with their biological inspiration. Another type of ANNs are spiking ANNs, which model the time-dependence of biological neurons and output discrete spikes rather than continuous values. However, these are less-suited for standard hardware and have been less developed and less popular than time-invariant ANNs.

2.1.2 Optimization of Weights

The goal of neural networks are to be able to perform well at a given task. In supervised learning, labeled training data is available, allowing the performance of a network to be quantified within the loss function of the network, measuring how far the model deviates from ground truth over the training dataset. Common loss functions include cross-entropy for discrete tasks like classification [25] or mean-squared error for continuous tasks like regression.

The synaptic weights can be optimized automatically through gradient descent of the loss function after random initialization of the weights [26]. Gradient descent has shaped many of the design decisions in ANNs, tending towards properties that are permissive to gradient descent such as differentiability. Other optimization approaches exist, but gradient descent has remained the most popular. Gradient



(a) A basic artificial neuron in layer $i+1$. The colored edges represent synaptic weight values.

(b) A basic fully-connected ANN with a single hidden layer.

Figure 1: Basic standard representations of a neuron and an ANN. The output of neuron j of the layer $i+1$ layer is computed by multiplying the activations of the ith layer, $\mathbf{x}_i$, by $\mathbf{w}_{i+1,j}$, summing the products together with the bias, and passing through the activation function, f.

descent tries to find the global minimum of a convex function, in this case the loss function of the ANN which measures the error of the network over the training data with respect to its parameters. In the space of this high-dimensional function, the gradient measures the change of the function value with respect to change of the parameters as a partial derivative. By subtracting the current gradient scaled by a learning rate parameter from the current parameter values, the next parameter values are generated for the gradient to be reevaluated at. This process continues repeatedly until convergence to an optimum, where the gradient will be zero.

Computing the gradient is accomplished via back-propagation through the layers of the ANN in reverse order [27]. While gradient descent is biologically plausible, back-propagation is much less so [28]. Although feedback connections exist in the brain, back-propagation requires precisely symmetric feedback for error communication and an explicitly known target at the final output of the network, neither of which have any biological evidence. How the brain solves this credit assignment is an ongoing research topic.

The most common basic variant of gradient descent is mini-batch stochastic gradient descent [26], where cycling mini-batches of data are used for each gradient calculation, as modern datasets are too large to efficiently compute the gradient across the entire dataset. Further improvements include momentum terms and adaptive learning rates to increase the speed of optimization and avoid problems such as local optima.

2.1.3 Convolutional Neural Networks

The simple ANNs discussed thus far are general networks that do not assume any symmetries in the data. However, more powerful and specialized ANN structures have been designed to improve performance on specific tasks. The most popular example is convolutional neural networks (CNNs) for computer vision tasks [9]. Images are stored in $H \times W \times 3$ format, with a height of H pixels, a

width of W pixels, and a channel depth of 3 color channels. A convolutional layer is parametrized by filters that are convolved over the image, multiplied element-wise over a local patch and summed for each translation of the filter. Each channel in a convolutional layer can be abstractly equated to a neuron in a standard feed-forward layer. Each channel in each layer is connected to all channels in the layer before and all channels in the layer after. Each connection is parametrized by a filter, which is a small $k \times k$ matrix of values, rather than a single value. CNNs are a special case of ANNs that could be equivalently represented with shared weights for relative local connections and null weights for all other connections. Convolutional layers are equivariant to translation, allowing CNNs to recognize entities in an image invariant of position.

CNN architectures often begin with convolutional layers that gradually increase the channel count while decreasing the spatial dimensions. For classification and similar tasks, the tail of the network will be feed-forward layers to make the final classification decision. Additional layer types often employed throughout the architecture are normalization layers and pooling layers. Normalization layers normalize activations across batches, with the intuitive benefit of avoiding overfitting to the training data distribution. Pooling layers perform downsampling by reducing the spatial dimensions and keeping a summary statistic of the original values such as the maximum or the average.

Further innovations in CNN architectures have potentiated their progress. One of the simplest yet most effective is skip-connections, or connecting the activations of one layer directly to the layer after the next one, thus creating a sum of the outputs of the two prior layers [29]. This is theorized to stabilise gradient descent, avoiding the problem of exploding and vanishing gradients, allowing CNNs to become incredibly deep.

2.1.4 Architecture Design

Until the last several years, architectures of deep ANNs were always hand-designed and generally static during the learning process. This provides a consistent basis for gradient descent: computers are better than humans at numerical tasks like optimizing parameter values of a given function such as a neural network, while humans are better at more abstract tasks such as designing the structure of the network. However, the desire to automate network design is increasing as computers become more powerful and human engineering is bottle-necking the potential of ANNs. This leaves formulating ANN design as a more computationally-suited problem as an open challenge.

Applicable to this challenge, there are many related sub-communities tackling various flavors of neural network optimization beyond parameter tuning. Meta-learning, or learning how to learn, aims to improve the learning process itself of an ANN [30]. AutoML tries to automate as much of the full machine learning pipeline as possible, including but not limited to hyperparameter optimization, learning algorithm selection, data preparation, and architecture selection [31].

Structural learning directly tackles this challenge of automated structural design. We focus on one-shot structural learning algorithms, analogous to the development of an individual brain over its lifetime. Thus, the optimization of parameters is performed in conjunction with structural optimization with the final goal of an employable ANN that is structured and parameterized specifically for a given task.

Next, we detail original inspirations particularly of the structural learning processes in the brain to provide a biological context for the rest of this report.

2.2 Biological Neural Networks

Although ANNs were originally inspired by the brain, their characteristics have diverged. The basics of neuron anatomy and function are briefly described here before focusing on the structural dynamics of biological neural networks.

2.2.1 The Basic Units of Biological Neural Networks

The quintessential unit in the brain is the neuron [32]. This cell is characterized by its dendrites receiving incoming information and the axon sending information to other cells via their dendrites. A single physical connection between two neurons is called a synapse. Mutable properties that affect how two neurons communicate include synapse connection strength, number of synapses, synapse location, and neuromodulators within a synapse. Neurons propagate signals via action potentials: when the cell membrane is activated enough via the dendrites, an action potential fires from the dendrites through the cell body and down the axon to outgoing synapses. Thus, neurons communicate in a time-dependent, spiking regime.

Although neurons are the most well-known cell in the brain, numerous other cell types support and potentiate their function. Glial cells alone outnumber the number of neurons in the human brain by tenfold [33]. They serves many purposes: oligodendrocytes provide myelination that increase speed and reduce loss of action potential transmission over axons, astrocytes modulate synaptic connections and support nutritional needs of neurons, and microglia prune synaptic connections. Neurons and their supporting non-neuronal cells dynamically cooperate to form the biological neural network, capable of the highest form of intelligence we know of today.

Biological networks have much more organic architectures that ANNs, but still show some structure and regularity on various scales [34, 35]. For example, much of cortex is organized into six layers with distinct physical and functional characteristics on each layer [36]. However, a key difference is that the brain structures itself over the lifetime of an individual [37], while ANNs are hand-designed and often have static architectures through the learning process.

Many structural learning algorithms for ANNs are inspired by biological neural development, from synaptic pruning [38] to neurogenesis [39]. To better understand the role of and possibilities for structural learning in ANNs, we investigate the roles of neuron creation, neuron removal, synaptogenesis, and synaptic pruning in the brain. Neural development in biological nervous systems and the brain is a complex process and a comparison with artificial representations necessarily requires simplification, but even a simplified view of biological structural learning offers insight for the artificial process.

We focus on the epigenetic timescale of neural development as it occurs in individual organisms in two distinct stages: early development and structural plasticity in the mature brain. Both have been used as inspiration for structural learning; for example, the self-organizing ANNs in [40] are based on early development while the synapse modulation method proposed in [41] is inspired by learning over the human lifespan. Both periods are relevant for structural learning in ANNs, offering different views on the role of neurogenesis, synaptogenesis, synaptic pruning, and neural cell death.

2.2.2 Neurogenesis and Neural Migration

Neurogenesis is abundant and rapid during early development. Starting from around 5 weeks of embryonic development, the human brain will develop almost all of the roughly 86 billion neurons present in the adult brain [42]. After approximately 28 weeks of gestation, this vast neurogenesis slows down considerably, with only a few regions continuing neurogenesis after birth [43, 44].

As this massive neurogenesis is taking place, cells also begin to move to specific locations, organizing into distinct regions and specific structures [45]. Chemical and sensory cues guide this neural migration, which is essential for the functioning of the brain. Neurons which started in distinct germinal zones may travel through specific complex routes to reach their final destination, resulting in different composition of neural types in different sections of the brain. By 18 months of age, much of the structure of the brain will be set by early neurogenesis and neural migration [46]. Structures like the cortical columns, which inspired convolutional neural networks, result from this neurogenesis and neural migration of early development [7].

Neurogenesis becomes much less common once early development is over; indeed, it was previously believed that there were no new neurons added to the adult human brain. There is growing evidence that neurogenesis occurs in the adult brain [47, 48], but has only been identified in specific regions. While the mechanisms behind adult neurogenesis are not yet fully understood, nearby neural activity has been identified as a factor, supporting the idea that the brain can create new neurons in response to needs which are not fulfilled by existing neural circuitry [49].

Beyond their movement and connection to other neurons, neurons are also capable of change during their lifetime. Learning can produce nonsynaptic changes of neural membranes, such as membrane conductances which modulate excitability [50]. While artificial neurons already modulate their bias during conventional, non-structural learning, these changes to the neural cell body could be seen as a more drastic change to the neural activation function.

From an artificial perspective, neurogenesis and neural migration can be considered as a single event in modern artificial neural networks, which rarely consider the physical placement of neurons. The neurogenesis of early development is therefore akin to neural architecture search space definition; the fundamental components of learning are in place but not yet properly wired. Adult neurogenesis in the biological brain, on the other hand, takes place in specific sections where existing structures and well-defined activity patterns abound. An artificial analogy of adult neurogenesis could be an expansion or modification of the possible synaptic weight space, such as the addition of a new layer, convolutional filter, or individual neuron.

Early development sets the stage for cognition and learning in the mature brain. Neurogenesis on a massive, rapid scale, with new neurons being created, differentiating, and migrating, is a hallmark of early development and is relatively unique to this period. Synaptogenesis next takes this stage in early development, with axons branching through complex routes to create the immense connectome of the brain and nervous system.

2.2.3 Axon Migration and Synaptogenesis

During early development, as new neurons are created and migrate to specific sections, connections between these new neurons begin to establish the neural circuitry of the brain. Shortly after neurons differentiate, genes which encode proteins necessary for synaptogenesis are activated [51]. The creation of connections between neurons begins during development but continues well into early postnatal life and is thought to contribute significantly to learning and memory in adults [51].

After differentiation, neurons project axons which then also migrate, following paths predetermined by genes or dynamically shaped by the chemical gradient trail of netrins released by other cells [52]. For almost all types of neurons, their singular axon will serve as the output line of communication, carrying an electrical discharge to other neurons. Axon development and migration happen throughout early development, with some axons traversing long and complex routes; the longest in humans is the sciatic nerve, reaching over a meter from the base of the spinal column to the foot.

In the central nervous system, innumerable connections form between neurons in intricate patterns during early development. As a neuron extends its axon, it will also extend many other branches around its cell body: dendrites, which receive electrical stimulation from other neurons. Once a searching axon of one neuron reaches the dendrites of another, synapses, individual connections, may be created. The formation of an individual connection, synaptogenesis, is regulated by chemical signals, activity of both the presynaptic and postsynaptic neurons, and genetic factors [46].

In early development, as in the mature brain, nearby axons and dendrites will exchange a number of signaling molecules which, in tandem with other cues, make neurons more receptive to synaptogenesis. The fate of newly formed synapses will be decided by the activity of the connected neurons, determining if they will be stabilized or eliminated [51]. A single neuron will form many similar connections which compete for stability as failed synapses close and the axon or dendrites retract.

While genetic cues inform many of the initial paths of axon migration and branch, neural activity sculpts neuronal arbor growth and synapse formation. Disruptions to neural activity have been shown to drastically change connectome organization, from the level of individual synapse formation to entire axon branch placement [53]. In developmental synaptogenesis, neural activity is a contributing factor amongst others for synaptogenesis, but as the brain matures, activity becomes an important trigger for further synaptogenesis.

In the mature brain, synaptogenesis plays a central role in associative learning and memory. Pharmaceutical treatments such as antidepressants have been shown to produce significant increases in the number of synapses in the adult brain [54]. New neurons in the mature brain may develop connections in a different way than neonatal neurons, first developing input synapses in their proximal dendritic domain before firing action potentials [55]. In this way, the connections formed by new neurons in the mature brain may be informed by existing neural circuitry before adding new connections and functionality.

From an artificial perspective, the synaptogenesis of early development can be considered as similar to developmental neurogenesis; the preliminary neural circuitry is defined largely by genetic cues which create the possible space for future learning through synaptic tuning. While the synaptogenesis of early development can depend on activity, it has also been demonstrated in the absence of activity, pointing to circuitry designs predetermined by genes [51]. In the mature brain, however, activity is an integral part of the integration of new neurons and synapses into the neural circuitry. New neurons will listen to nearby activity before activating, forming downstream synapses based on observed activity patterns. As with neurogenesis, synaptogenesis sets the stage for learning to take place, but it is with the stabilization and elimination of parts of this neural circuitry where specific learned patterns in the brain are shaped.

2.2.4 Synaptic Pruning

Connectivity patterns in the mature brain are learned through a process of correlation detection, synaptic competition, and eventual synapse elimination, i.e. pruning. The pruning process has been identified as a crucial component of learning and of typical neural functionality. It is widespread in the central nervous system but has also been observed in the peripheral nervous system [56]. Pruning occurs at a massive scale during development but continues throughout lifetime.

The prevalence of pruning in the developmental and adult brain indicates the presence of numerous redundant or excess connections formed during development. The reasons for these exuberant connections are not fully clear, but there are multiple possibilities, one being physiological or structural roles of these excess connections during early development [56]. More importantly from an artificial perspective, however, is the information available for neural circuit determination. During neurogenesis and synaptogenesis, possible communication between cells is limited to chemical signals and a limited number of connections. Excess connections offer a rich information pipeline for neural circuits to refine into specific functional patterns.

The information pipeline of highly-connected neural circuits offers a way to detect correlation between connected neurons. Experimental evidence has demonstrated that neural activity regulates synapse elimination [51], showing that correlated synaptic activity informs about the relevance of synaptic connections and influences synapse stability and elimination as postulated by Hebbian theory [57]. Neurons with high levels of correlated activity will retain connections while others are pruned, forming specific circuits based on activity patterns.

The development of the visual system demonstrates the importance of activity for eventual circuit formation. Highly correlated patterns of spontaneous activity in the form of retinal waves occur throughout development, driving activity-dependent pruning in the visual system [58]. Experiments in ocular deprivation have shown that a deprivation of activity in one eye can lead to structural changes

in the visual cortex which result in a lack of response to new activity in the covered eye [59]. The brain undergoes significant rewiring in response to specific novel activity patterns, in the visual cortex and elsewhere.

Beyond the local information of presynaptic and postsynaptic neural activity, other signaling mechanisms influence synapse elimination. Microglia in particular have been identified as an important mediators which promote and constrain synaptic elimination [60]. The microglia can tag unwanted synapses with specific proteins leading to their elimination [61]. While the full internal mechanisms of the glial cell tagging decision are not yet completely understood, the molecular mechanisms involved in synapse elimination are spatially and temporally dependent, and disruption to these mechanisms has been linked to brain dysfunction [56]. From an artificial perspective, this motivates the design of pruning decisions which are made not only based on internal factors such as synaptic weight and activity information, but through complex and dynamic policies which account for various information over entire regions of the brain.

Active neurons can have a large number of synapses pruned with little effect on the neural cell body. However, in the event of pruning along an entire axon branch, the branch will retract or be eliminated from the cell [62]. Retrograde signaling mechanisms pass information from distant axon heads back to a neuron cell body to inform of pruning events and to initiate an appropriate response [63]. This may involve restructuring of the axon or dendritic tree of the neuron, but can also in some cases elicit neural cell death. In this way, sufficient synaptic pruning and appropriate signaling can lead to complete neuron removal, as we examine in the next section.

2.2.5 Neural Cell Death

Neural cell death can be distinguished into two main types based on timing: a common early cell death occurring in neural progenitors and immature neurons, and a late cell death which occurs in postmitotic neurons [64]. There are many mechanisms and underlying causes for neural cell death, with some mechanisms shared between the two types, but in many ways, they are distinct processes. We consider both types, aiming to clarify the decisions leading to cell death in both cases.

Early cell death is a necessary part of neural development and occurs either before full differentiation of the neuron and its projections or during synaptogenesis [65]. This process is a mechanism for cell number and quality control, removing unwanted cells and those damaged by proliferation. While this contributes to the final neural circuit through cell number control, experiments on the prevention of neuronal cell death during development have demonstrated that phenotypically similar circuitry can arise despite the presence of excess neurons [66]. However, neural cell death does regulate mature functional circuitry through controlling competition over factors necessary for cell survival such as chemical signals from glial cells [65]. Notably, early neurons require a constant signal to prevent cell death; in this way, elimination can be seen as the default end of new neurons with persistence as the exception.

Early neural cell death is abundant in early development, in tandem with the massive amounts of neurogenesis, and it also accompanies adult neurogenesis. Adult neurogenesis has been observed in the olfactory bulb but from the new neurons generated, only 50% will survive for extended periods of time and be integrated into the existing circuitry. Interesting, in this case, neural activity was demonstrated to be am important factor in determining cell survival [55].

From an artificial perspective, inspiration from biological early neural cell death can inform the decision of elimination of new or untrained operators such as layers. As the parameters of these operators are not yet informed, evaluating their potential for removal can be difficult yet rewarding. The biological process of creating a large excess of cells and eliminating quickly can serve as motivation for artificial structural learning.

Programmed cell death in mature and connected neurons is a much more rare event in healthy brains. The main causes of postmitotic neural cell death are the death of other connected neurons (transneuronal degeneration), axon degeneration (Wallerian degeneration), or the result of various signals from other neurons and glial cells [67]. In the adult brain, these events typically occur in response to an injury or as part of a neurogenerative disease. Transneuronal degeneration occurs relatively frequently in young animals where new neurons are still being formed, but is much less common in adults. Wallerian degeneration mostly occurs in response to axon injury, inflammation, or genetic impairment. The signals which can elicit mature cell death are numerous but are most common in neurogenerative diseases such as Alzheimer's disease [67].

As the death of highly-connected, postmitotic neurons is such a rare event in healthy brains, the main contribution of neural cell death to the learning process occurs early on, during or even before synaptogenesis. This can result in a chain of cell death, transneuronal degeneration, but this event becomes rarer as neural circuitry matures. This indicates that, as opposed to individual synapses which are often pruned throughout learning, neurons are costly to remove. While this principle correlates well to highly parameterized ANN operators, it is much less costly to remove an artificial neuron than a real one, as the costs of cell removal and replacement neurogenesis are greatly reduced.

2.2.6 From Biological to Artificial

It is a drastic simplification to summarize these complex processes in the brain, which are still not fully understood, with the four previously defined structural learning operators. However, artificial neural networks have different information and energy requirements than biological neurons. Artificial neurons which are represented as distant can be connected just as easily as neurons which are represented as adjacent. Information is transferred between artificial neurons without loss; synapses don't have to be protected from possible interference. There is no bandwidth limitation for individual synapses nor additional information gained from synapse location, so the relationship between presynaptic and postsynaptic neurons can be summarized by a singular continuously-valued synaptic efficacy measure. The similarities and differences between biological neurons and the artificial models inspired by them are constantly evolving and being better understood, but it is clear that their fundamental differences require different learning operator definitions.

The result of these simplifications is that many decisions made by cells in the biological brain are aggregated into the structural learning operators. The variety of information sources for cellular decisions are streamlined in ANN operators; chemical signaling is mostly ignored in ANNs as is cellular contact. However, recent trends in ANN literature have begun to explicitly reintroduce the sort of information pathways used in the biological brain, such as the role of non-neuronal cells [42], Hebbian synaptogenesis [68], and activity-dependent neuron removal [69]. Understanding the complexity of these structural changes in the biological brain can bring new perspectives on the formulation of structural learning operators in ANNs.

3 Analysis of Structural Learning

Structural learning is the optimization of both the architecture of an ANN and the parameters of that architecture in a single process. As shown in the following work, most of this form of structural learning used in practice has taken the forms of pruning, or the removal of parameters to reduce network size while maintaining performance, and Neural Architecture Search (NAS), or the automated search of architectures within a search space. Relatively few papers have pursued developmental structural learning, which involves creation of new structures or connections within the network.

In this review, we take a biologically-inspired view of structural learning in ANNs. The latest analysis of structural learning in both biological neural networks and ANNs was completed more than two decades ago [70]. Since then, there have been many advances in both artificial intelligence and neuroscience research, prompting the need for a modern analysis. These links have already been well-defined for the neural and biological modeling communities, but there is still a disconnection between these communities and deep-learning with ANNs.

In an effort to surpass the structure design bottleneck, we look to the parallels of ANNs in biological nervous systems in order to define a framework for structural learning in Section 3.1. We survey methods of structural learning in ANNs already existing in literature, providing broad statistical overviews in Section 3.2 as well as detailed tables of features over our selected corpus at the end of this chapter, then diving deeper into each operator of our framework across implementations in Sections 3.3 and 3.4. We finally give suggestions on how to progress forward on current challenges in Section 3.5.

The aim of this work is to shed light on a set of similar research directions which are already well established in disjoint fields but not yet often linked. Using a common language for structural learning may help connect the many subcommunities of ANN research that are already researching similar approaches to structural learning with different implementations, abstractions, and goals.

This work is heavily based on Maile et al. [16]. Recent related works within the scope of this survey may not be included after its original publication.

3.1 Framework Definitions

In order to understand structural learning, we propose a definition which, in this work, will be used to define the scope of study. We consider structural learning to be a change to a neural network during its life-cycle through any of the following four atomic operators, referred to subsequently as the **neural operators**:

- **Neuron creation**: the addition of a unit to an ANN,
- **Neuron removal**: the removal of a unit from an ANN,
- **Synaptogenesis**: the creation of a non-zero weight between two units,
- **Synaptic pruning**: the removal, or change to zero, of a non-zero weight between two units.

We purposefully open the definition of neuron creation and removal to different unit types: in many structural learning algorithms, entire groups or “units” of parameters are added and removed together. Units may be as simple as a neuron in an ANN, or be another higher level structure such as a convolutional filter, channel, layer, or module of layers. A unit contains a portion of the parameters within the network and provides basic organization to make the construction of a network more tractable. While

the basic computational units in both the brain and in ANNs are conventionally called neurons, we use unit as a more generic term that can include any structure within a network being used as nodes in a graphical representation of the structure.

The terminology used in this framework is not yet adopted in structural learning literature. For example, what we define as neuron creation is often referred to as neurogenesis, although the latter process is much more involved in the brain than in the artificial case as discussed in Section 2.2.2. The differences between existing and often inconsistently used terms and our own will be discussed subsequently.

The dominant form of learning in ANNs is weight training through error backpropagation, which optimizes synaptic weights and neuron biases in an objective function landscape using gradient descent while keeping the other characteristics of each neuron constant: the layer type, activation function, and pattern of non-zero connections. In this chapter, we aim to characterize the impact of these neural operators on learning, which permit changes to connectivity and are often coupled with weight training in a bilevel optimization process.

To understand the role of these neural operators in the context between biological and artificial neural networks, we next review relevant implementations of them in existing ANN literature.

3.2 Characterizing Structural Learning in ANNs

Using our neural operators of neural creation, synaptogenesis, neural pruning, and synaptic pruning allows for characterization of these algorithms from a structural learning perspective, drawing links between different implementations of the neural operators. Some instances of these neural operators are brain-inspired by design, while others are incidentally correlated to biological operations.

In general, ANNs are constructed as a composition of individual components, layers, in a directed and usually acyclic graph. We refer readers to Goodfellow et al. [23] for a full review of the different basic components of modern ANNs, such as convolutional, pooling, and normalization layers.

3.2.1 Scope of Study

Our 59 collected papers from ANN literature each demonstrate at least one of the four neural operators, neural creation, synaptogenesis, neural pruning, and synaptic pruning, over the course of training an individual ANN via an optimization method. We focus on standard ANNs with parameterized connections, generally with feed-forward, convolutional, or recurrent layers.

Our focus on standard ANNs excludes time-dependent networks, like spiking neural networks. While more biologically inspired, these have not been as powerful nor as common as time-independent ANNs so far on standard computational hardware.

ANNs simplify the biological complexity of connection strength between two neurons into a single continuous value. We consider all non-binary changes to this weight, which could represent biological structural change, to be weight training, not structural learning. This includes soft structural methods that do not make any discrete changes to the architecture and thus no changes to the effective dimensionality of the objective function. Only hard structural methods are evaluated, including those which involve a discretization step after soft structural learning, such as Liu et al. [19].

While synaptogenesis necessarily follows neurogenesis in the brain through different processes, we consider the non-learned creation of connections of a new unit to be encapsulated within neuron creation in ANNs, such as adding a neuron with default connections to all neurons in the previous and subsequent layers. Thus, in our characterization, algorithms are only labeled with both neuron creation and synaptogenesis if there is a connection learning component for existing units in addition to a process for creating new units with a default set of connections.

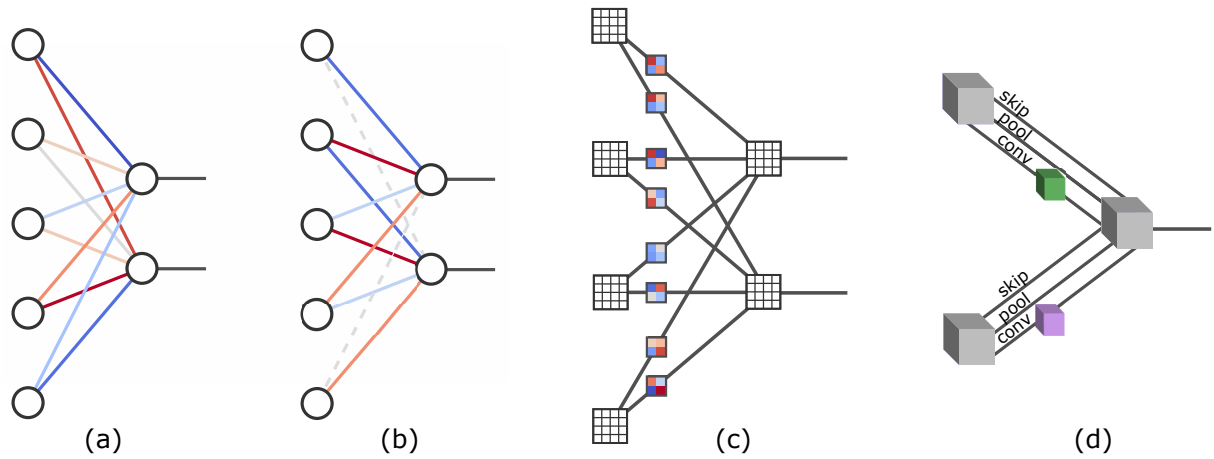


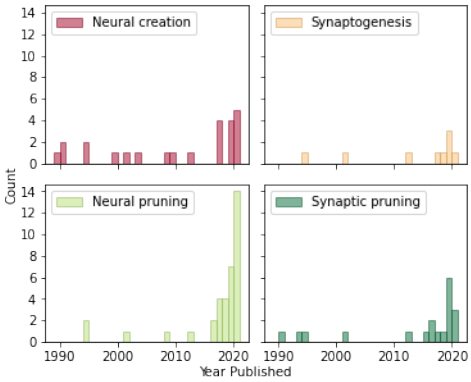
Figure 2: Levels of graph abstractions found in structural learning algorithms. From the level of **(a)** feed-forward neurons, there are the two levels of convolutional paradigms: **(b)** on the same level as FFNNs with weight-sharing and systematically pruned connections, and **(c)** on a higher level with a parameter matrix on each synapse and an activation matrix within each neuron, compared to singular values respectively on the lower level. The shared weights in **(b)** are referred to as a kernel, which are the small matrices parameterizing each edge in the higher level paradigm in **(c)**. On an even higher level is the NAS super-network paradigm **(d)**, where each edge may contain parameters based on the operation type, like the parameter-less skip-connection and pooling and the parameterized convolution operations shown. Higher levels have higher dimensionality, structure, and complexity within each node and edge.

Many structural search papers enumerate all possible options within their prescribed search space throughout the learning process in order to select the best combination of structures to build the final architecture. For example, some NAS algorithms initialize all possible layers with all possible connections and learn which ones are best. We consider this as entity pruning, rather than entity creation.

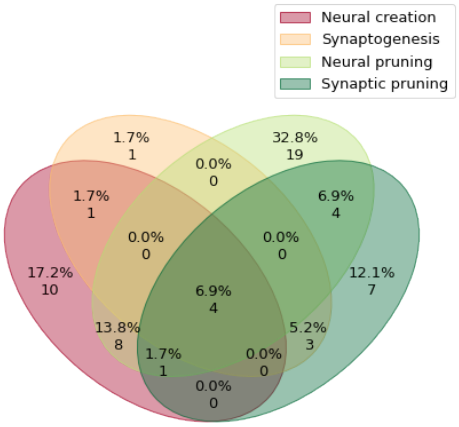
A biologically inspired family of algorithms for discovering ANN structures automatically are evolutionary algorithms. These algorithms model biological evolution by maintaining a population and recombining the genes, which may directly or indirectly encode architectures, of successful candidates to produce more candidates. Some such methods meet our criteria for structural learning by persisting network weights across individuals within the population [71, 72]. However, many evolutionary methods maintain a population of individuals with different structures and weights, such as NSGA-Net [73] and NEAT [74]. These methods include structural learning operators; in NEAT, for example, new neurons or connections can be added through mutation. However, to compare with structural learning in single networks, changes through mutation from one generation to another would have to be studied on specific individuals. We further discuss evolutionary methods in Section 3.5, but in the following sections, we only include search methods that which persist weights and study structural changes on individual networks. This limits the collected corpus to one-shot algorithms, which pursue a more continuous path in the changing objective function space during the co-optimization of architecture and parameters.

We exclude architectures with self-gating mechanisms, such as LSTMs [75] and transformers [76], from this framework and corpus. These neural networks change their own structure in a transient and input-dependent manner, occurring on a shorter timescale than structural learning. This is more akin to neural circuitry gating and modulation [77]. We discuss such dynamic architectures further in Section 3.5.2.

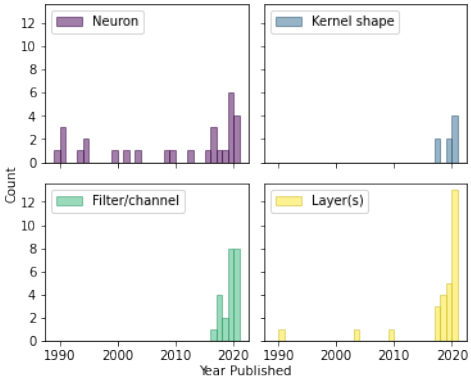
The abstraction from individual biological neurons to artificial feed-forward neurons, the most basic type, is clear. However, convolutional neural networks (CNNs) have two levels of abstraction, depicted in Figure 2. The first is by considering a CNN as a special case of a feed-forward neural network



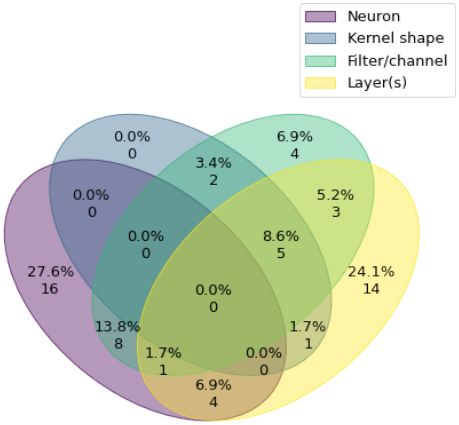
(a) Occurrence of operators over time.



(b) Co-occurrence of operators within the collected corpus.



(c) Occurrence of unit type over time.



(d) Co-occurrence of unit type within the collected corpus.

Figure 3: Characterization of neural operators and unit type used in our selected corpus of papers.

(FFNN) with non-local weights zeroed out and local weights shared. The other paradigm moves the neuronal unit up to a higher level, considering a channel in each convolutional layer to be the neuron abstraction. Connections between channels represent synapses and are parameterized by a small matrix kernel rather than a single value, and the activation within the neuron is a larger matrix rather than a single value. While the lower level paradigm continues to multiply the singular activation coming as input to each synapse by the single shared synaptic weight value, the higher level paradigm performs a convolution of the kernel parameter matrix over the incoming activation matrix before summation across connections. Using the lower level paradigm of Figure 2b for structural learning enables partial-area convolution or changing kernel characteristics such as size and striding across neurons and is denoted as "Kernel shape" in Table 1. Using the higher level paradigm of Figure 2c becomes adding and removing channels as neurogenesis and neural pruning, respectively, denoted as "Filter/channel" in Table 1. An even higher level paradigm is super-networks in NAS, shown in Figure 2d, denoted as "Layer(s)" in Table 1. We consider works on structural learning which function on these different levels of abstraction.

3.2.2 Characterization of the Collected Corpus

The goal of this work is to characterize structural learning, rather than document all uses of any of the four neural operators in the vast ANN literature. Our selection of 59 seminal works is therefore not comprehensive and is rather intended to be a qualitatively representative sample, as there are

orders of magnitude more papers in the subfields of NAS and pruning than in the developmental structural learning space. Completing a more rigorous collection process for complete quantitative characterization would be prohibitively nuanced, for reasons including inconsistent vocabularies surrounding structural learning used across subfields and subtle differences between novel structural algorithms versus applications of existing techniques to new architecture types and task domains, and thus is beyond our scope. The 59 works in the collected corpus provide a diverse and informed view of the structural learning literature.

We refer the reader to other reviews for a more complete view of specific domains. An overview of early works in structural learning and their comparison to biological learning is presented in Quinlan [70]. More recently, Deng et al. [78] covers pruning and many other methods for model compression. Hoefler et al. [79] discusses sparsity in neural networks, which is usually achieved via pruning. Elsken et al. [80] describes the state of NAS as a burgeoning sub-field, while Xie et al. [81] is more recent and focuses on weight-sharing NAS. He et al. [31] covers AutoML, which encompasses methods that contribute to automating the machine learning pipeline beyond parameter tuning and thus includes structural learning as well as data preparation, hyperparameter optimization, and other types of architecture search. Stanley et al. [82] gives an overview of the use of evolution for neural network optimization, including methods which change neural structure over generations. Evolutionary developmental methods which evolve rules for structural learning are reviewed in Miller [83]. Parisi et al. [84] discusses methods for continual learning in ANNs, many of which use forms of structural learning to increase the information capacity of networks performing multiple tasks. Han et al. [85] covers dynamic ANNs, considering adaptability of the network beyond just connectivity during training as in our framework. Our goal in this chapter is to offer a coherent synthesis of works from these different domains under the framework of structural learning.

The general statistics for neural operators and units across the collected corpus are shown in Figure 3. These plots show trends and frequencies of characteristics across the collected corpus over time, although not necessarily representative of structural learning without the aforementioned biases in our sample collection. Neural creation and synaptic pruning appeared earlier, while synaptogenesis and neural pruning have become more popular in recent years, shown in Figure 3a. Most papers perform a single operator, but a significant portion perform a combination, shown in Figure 3b. Operating on units of standard neurons has been consistently used over time, while the arrival of deep learning architectures is evident in Figure 3c with structural learning over layers and convolutional units of kernel shape and channels becoming more popular after 2010. The impact of deep learning is also evident in Figure 3, with a significant increase in pruning methods while the automatic creation of new units or connections did not undergo a similar expansion of interest. We expect this is partially due to the relative complexity of the creation decision, which we explore in Section 3.4. Finally, as with neural operators, operating on a single unit type is common, but more algorithms are permitting structural learning across different units and layer types, shown in Figure 3d.

Further trends over the qualities of each algorithm in the collected corpus sorted by publication year are detailed in Tables 1-2 at the end of the chapter. The earliest algorithms were designed before convolutions and recurrent layers, but now most algorithms are designed for and demonstrated on ANNs with convolutional layers, with some additionally applied to dense layers or recurrent layers. Regarding tasks, nearly all algorithms are demonstrated with supervised learning tasks, particularly image classification in more recent years. Some specific techniques are applied to multi-task, continual learning, more specialized computer vision, natural language processing, sequential data, and reinforcement learning tasks: all of these pose additional challenges above the relatively simple image classification or tabular dataset classification baselines. We discuss existing structural learning methods in the context of each operator in Sections 3.3-3.4 and relevant implementation trends in Section 3.5.1.

Each of the four neural operators have different considerations and effects within structural learning. Performing synaptic pruning or neuron removal on an ANN is more tractable than their additive

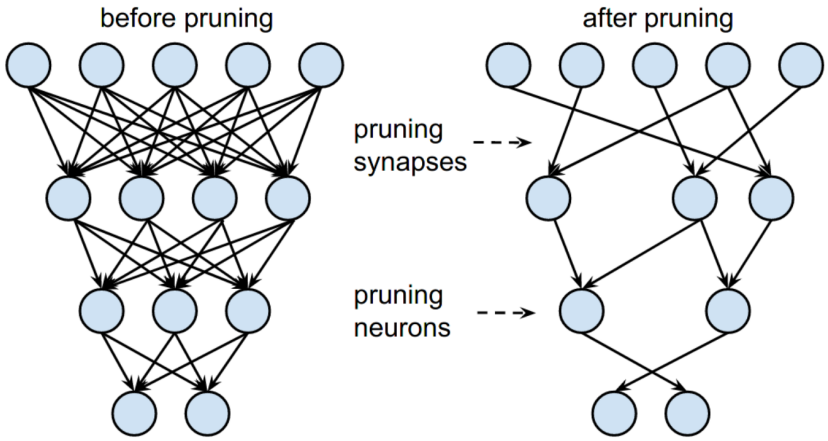


Figure 4: Neural pruning and synaptic pruning in a fully connected ANN [86]. When a neuron is pruned, as in structured pruning, all incoming and outgoing synapses are also removed. Synaptic pruning additionally removes synapses between unpruned neurons and is also referred to as unstructured pruning.

counterparts, because their structural learning search space is naturally defined by the existing network structure rather than being open-ended. As the two pruning operators have also been more commonly used in literature, we will consider them first in Section 3.3 before discussing the two creation operators in Section 3.4.

3.3 Synaptic Pruning and Neural Pruning

Our main questions to characterize pruning in our selected papers are:

- **Goals of Pruning:** Why remove connections or units from an ANN?
- **Biological Inspiration:** How can biological synaptic pruning and programmed neural cell death inspire artificial methods?
- **Architecture Specificity:** How has pruning been adapted to different ANN architectures, such as dense layers and convolutional layers?
- **Measuring Entities:** How is the search performed? How is importance of a connection or unit measured in order to make the pruning selection?
- **Scheduling Pruning:** When and how often does pruning occur with respect to optimization of the unpruned parameters?
- **Handling Pruned Entities:** How should pruned connections and units be handled?
- **Impact on Weight Optimization:** What is the impact of pruning on optimization of parameter weights?

The following subsections consider the implementations of synaptic pruning and neural pruning found within our corpus. We will first discuss a subset of these questions for pruning in general in Section 3.3.1 and then dive further into specifics of all questions for synaptic pruning in Section 3.3.2 then neural pruning in Section 3.3.3. Finally, we will draw conclusions and connect our discussion back to biology in Section 3.3.4.

3.3.1 Pruning Generalities

Pruning aims to remove parameters in an optimal manner, but their effect is not independent, and evaluating the effect of all possible combinations of parameters to prune on the network performance is not computationally feasible. Not only do the combinations grow exponentially with the size of the network, but the cost of completely evaluating each one by locating the new optimum through optimization to convergence is prohibitively expensive. Thus, algorithms generally make an approximation by estimating the effect of removing each parameter independently from the local landscape of the objective function on training data [79]. This naturally leads to greedy pruning decisions from the static set of weights, so parameters that currently seem beneficial to remove are selected at the time of pruning. Algorithms can only assume that this process approximates the globally optimal solution for what to remove from a network. Many Neural Architecture Search (NAS) papers formalize the difficulty of optimizing architectures and network weights simultaneously as a bilevel optimization problem, to be discussed further in Section 3.3.3.

Goals of Pruning: Pruning an ANN, or the removal or zeroing of portions of the network, to make it more efficient in space or time while maintaining performance is one of the many orthogonal ways to perform network compression. Using smaller networks with the same performance is particularly desirable for real-time applications and resource-limited devices such as mobile. Pruning large, over-parametrized networks to find their best-performing sub-networks is also a method for finding a high-performing architecture for a given network size. These naturally lead to applications of pruning in multi-task learning, where portions of the network are shared between tasks and others are task-specific, and NAS, where the architecture is optimized on a higher level.

Beyond efficiency, pruning can also yield generalization benefits [79, 87]. Due to over-parameterization of the base model, pruning can remove learned noise. This is why many pruning schemes show slightly increased test performance at low to moderate pruning levels. However, the benefit only applies within the initial distribution: pruning may come with a cost of reduced robustness to distribution shifts, but this may be ameliorated by explicit regularization [88].

Biological Inspiration: Activity-based synaptic pruning is a fundamental component of structuring the information circuitry of the biological brain, integral in varied processes such as memory formation and motor skill learning [51, 89]. Similar to ANN pruning methods [38, 90, 91], synaptic pruning in the adult brain uses measures of information passage in synapses to remove redundant connections and form sparse functional patterns. Less common is neuron removal, which is most prevalent during early development when neuron removal is the default for new neurons and only those with useful activation patterns survive to maturity [42]. In both cases, the brain relies on the creation of an excess of neural circuitry which is refined into the pertinent functional circuitry through pruning and removal.

Measuring Entities: Each entity must be evaluated in order to make an informed pruning selection. The final effect of changing an entity at a given training step cannot be predicted. Entities to prune may be selected by a salience metric computed per-entity designed to estimate the effect of removing the entity on the performance. Masks can also be used, introducing a new parameter for each entity that controls whether it is active or not. This is particularly useful for ephemeral pruning, where pruned entities may be reactivated later in the course of learning. The mask may be discretely controlled by a salience metric or relaxed to continuous values that can be trained via optimization. Finally, regularization can be used to encourage sparsity of network or mask parameters during gradient descent steps by adding a magnitude-penalizing term to the objective function.

Once the selection metric or mask is evaluated, the algorithm may then use either ranking or thresholding, at either a local or global scale, to discretize the pruning decision for each parameter. Ranking allows a defined proportion to be pruned, which may be particularly desirable for hard time or space constraints of the final ANN, while thresholding allows for only parameters meeting a score criteria to be pruned, which can be more consistent over iterative pruning phases. Local pruning makes the ranking or thresholding decision within a structure such as a layer, which makes comparisons more

homogeneous, while global pruning makes these comparisons over the entire architecture, which allows the algorithm to determine the pruning level for each structure automatically.

Scheduling Pruning: The schedule of pruning determines how it is interwoven with weight updates, including when and how often. Iterative pruning, rather than all-at-once, has generally been more effective [90]. It allows the network to evaluate entities in a more intermediate state of pruning, lessening the effects of the assumptions of pruning independence between parameters, high order terms ignored in most metrics, and local objective function evaluations. On the other hand, fully pre-training a network before pruning may be expensive but can be completed independently of the pruning, such as using an off-the-shelf pre-trained model. Most pruning algorithms begin with a large, dense network, while some perform dynamic sparse training, where the initial network is sparsely initialized [90, 92–94].

Handling Pruned Entities: After pruning, the pruned entities may be either permanently disregarded or allowed to be revived, which we name ephemeral pruning. We distinguish ephemeral pruning from entity creation if the revival is done with the same method as the pruning and thus is bounded to the same search space as pruning.

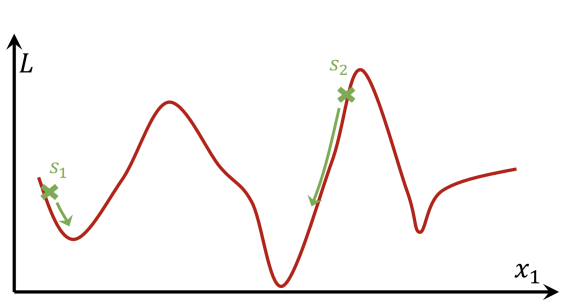
Impact on Weight Optimization: Because the objective of pruning is often to at least maintain performance, this is synonymous to reducing the dimensions of the objective function space as much as possible while maintaining or decreasing the cost of the found minimum upon convergence of the objective function. Convergence to a globally optimal point is not guaranteed for standard gradient descent techniques in structurally static networks, but the local minimum found usually has a low enough cost in practice [23]. Because pruning potentially changes the cost and location of each optimum, this is necessarily an even more difficult optimization than training a structurally static network. Techniques mentioned so far such as iterative pruning and regularization intuitively help ameliorate negative effects of the discrete changes in the shape of the objective function while performing gradient descent in parallel to pruning, thus aiding in the search [79].

Pruning may be categorized by the level of structure used in the pruning process, as shown in Figure 4. Unstructured pruning, which we discuss further in Section 3.3.2, is the zeroing of any parameters within an ANN and thus is generally considered synaptic pruning for our neural operator definitions. Structured pruning, which we discuss further in Section 3.3.3, is the removal of entire units and their associated parameters within a network, such as channels, neurons, or layers, so it is synonymous with neural pruning.

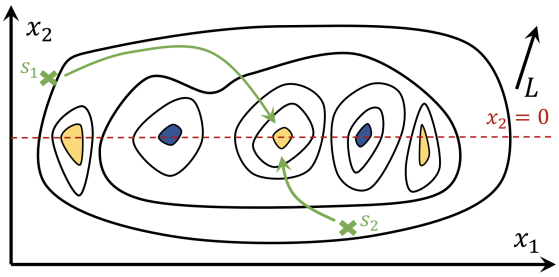
3.3.2 Pruning Connections

Goals of Pruning: Synaptic pruning is the lowest level and simplest form of pruning; it intends to remove parameters, which each represent a connection, without any pattern or structure. The earliest pruning papers began with unstructured pruning [38, 95]. Pruning synapses in a network is accomplished by zeroing connections and may allow a lower space complexity of storage by storing the remaining parameters in sparse matrices. However, it does not yield significant improvements in time complexity of training or inference without specialized software for accelerating sparse matrix multiplications [90]. ANNs after unstructured pruning can outperform dense models of the same memory footprint and architectural skeleton[96], showing that the pruned models can be more specialized and effective. The main applications of synaptic pruning are for network compression [38, 86, 90, 95, 97, 98] and for counterbalancing synaptogenesis [68, 93, 94, 99–101]. It also benefits multitask learning such as in Peng et al. [91], where only important connections are saved for the current task and the pruned connections are reused for future tasks.

Biological Inspiration: Synapses are continuously being generated and pruned in the adult brain as a part of learning [89]. Synaptic pruning in the biological brain can result in weight change, removal of synapses in otherwise connected neurons, or wiring changes, complete disconnection of two neurons



(a) Single parameter stochastic gradient descent, where continuing descent from s_1 results in a local minimum.



(b) Double parameter stochastic gradient descent, where continuing descent from either s_1 and s_2 results in descending to the global optimum.

Figure 5: Comparison of stochastic gradient descent in a toy loss landscape without and with an extra dimension (figure reproduced with permission from Hoefler et al. [79]). Transitioning from **(a)** to **(b)** represents creation, while transitioning from **(b)** to **(a)** represents pruning. s_1 and s_2 are potential parameter value(s) at the time of structural change.

[102]. The latter is the equivalent of ANN pruning, where continuous weight values are removed or set to zero. In the brain, this decision is based on activity correlation [57], where neurons with higher absolute correlation in their activity stay connected while other connections are pruned. This can be modeled as a locally-available measure of importance based on Fisher information [103], similar to unstructured pruning methods in ANNs. However, other decision factors in biological synaptic pruning could be further explored in ANNs. Glial cells have been shown to tag potential synapses for removal using chemical signals over regions of the brain, using spatially and temporally dependent tags [56, 60, 61]. This allows for coordinated pruning of multiple synapses on different neurons which share characteristics such as a response to certain activity patterns.

Architecture Specificity: Unstructured pruning is simple to apply to different neuron and layer types, since it can be agnostic to the layer type and just removes single parameters at a time. For convolutional layers, synaptic pruning can mean removing single values within a 2-D kernel [68, 86, 93, 97, 98, 100, 101] or the entire kernel itself [94]. Although each kernel contains more than a single parameter and is thus a higher structure, it can abstractly represent a connection between channels in convolutional layers, similar to a connection between neurons in dense layers [94], and cannot be explicitly removed from the 4-D convolutional filter matrix, only zeroed, so it is more akin to unstructured pruning.

Measuring Entities: Salience metrics try to estimate the importance of each connection in order to prune unimportant connections. They can be as simple as the absolute parameter value itself such that weights near zero are pruned [68, 86, 100], which is intuitive as they would have a lower impact on the ANN than weights with stronger magnitudes. More complex calculations include gradients, which measure sensitivity of the network outputs with respect to each parameter, so low values signify unimportance. Any metrics requiring a derivative of the objective function must be computed with respect to training data. For most modern datasets, computing them for the entire dataset is not feasible, so mini-batches are often used. A local estimate of the effect of removing a single parameter on the objective function may be derived using a Taylor expansion [104]. The earliest pruning methods often used up to the second order term, including a full Hessian matrix, within their saliency calculations [38, 95], while current methods usually use cheaper but less accurate approximations with at most first-order derivatives over a single mini-batch [91, 98, 101, 105]. For example, the diagonal Fisher information matrix is a first-order metric, scaling linearly with dimensions in computational complexity, that is often used as an approximation of the Hessian matrix, a second-order metric [91]. The choice of salience metric is usually a trade-off between efficiency and effectiveness, but exactly how useful the precision of saliency is may be confounded with the greedy, iterative pruning process

often used: such metrics can only measure local information without interdependencies, so using more expensive methods may be futile.

Masks are also utilized in synaptic pruning. The mask may either be relaxed to be continuous so it can be trained along with the other network weights through optimization [93, 94, 97, 106] or remain discretely binary and be controlled with metrics [68, 100].

Scheduling Pruning: Regarding the training portion of the schedule, pre-training the large unpruned network is a common first step to a pruning, but some synaptic pruning algorithms avoid this for training time or space efficiency. Lee et al. [98] forgoes the pre-training step and instead prunes a newly-initialized model using a first-order derivative metric of connection sensitivity. Hassibi et al. [95] determines the optimal weight update for the remaining parameters after each pruning, but the inverse Hessian required is too expensive to compute exactly for modern ANNs, although estimations can be used [79]. Most neuron-level synaptic pruning algorithms use fine-tuning optimization steps after pruning to arrive at the final architecture with optimized weights.

Handling Pruned Entities: Most metric-based synaptic pruning papers do not allow discrete synapse revival, as computing metrics for inactive parameters often is not possible. However, masked selection techniques are more amenable to ephemeral pruning of synapses, which may help ameliorate the negative effects of greediness. Dynamic connection search allows previously pruned connections to be revived if they show a resurgence in parameter importance later in training [93, 94, 97, 100].

Impact on Weight Optimization: Synaptic pruning may decrease the dimensionality of the objective function search as finely as a single dimension at a time. The proceeding effect conjectured by Hoefler et al. [79] is demonstrated in Figure 5. This artificial example supports performing more gradient descent in the higher dimensional space and pruning later in the course of training, at least for traditional pruning occurring intermittently during training. If the network in Figure 5b is initialized to s_1, pruning x_2 before the value of x_1 surpasses the coordinate of the local maximum will result in convergence to a local minimum, but pruning after should allow convergence to the global minimum. Pruning non-zero parameters results in a displacement in the search space based on how large in magnitude the pruned parameters were. Magnitude-based pruning has a minimal but not negligible effect on the displacement of the location in the search space, and may also be more greedy. The performance change due to this displacement does not have clear consequences: the final performance cannot be tractably predicted for any given structural operation. Bartoldson et al. [87] found that the immediate performance drop after an iterative pruning step was positively correlated with improved generalization and performance of the final network, while Laurent et al. [104] had conflicting findings, showing loosely inverse correlation. While the objective function is an intuitive tool for optimizing the pruning selection, the interacting effects of the many local assumptions and approximations as well as hyperparameters such as algorithmic details confound its use.

3.3.3 Pruning Units

Goals of Pruning: Neural pruning is the removal of units within an ANN, such as neurons in fully-connected layers, filters or channels in convolutional layers, or entire layers themselves. Thus, neural pruning has different selection and scheduling considerations compared to synaptic pruning. Structured pruning easily yields both time and space efficiency benefits: entire portions of the network can be removed, thus skipping their computational steps and storage requirements.

Beyond network compression, structured pruning is also a very common abstraction for performing Neural Architecture Search (NAS), an emerging sub-field pursuing the automation of architecture engineering at the layer level. ANN architectures are selected from a predefined search space of possible architectures and evaluated according to a performance estimation strategy. This performance estimation then informs the search, which can be based on evolutionary algorithms, reinforcement learning, gradient descent, or other search methods. Structural learning can be used for NAS by using any of the four neural operators as search operators on persistent networks. Some existing NAS methods

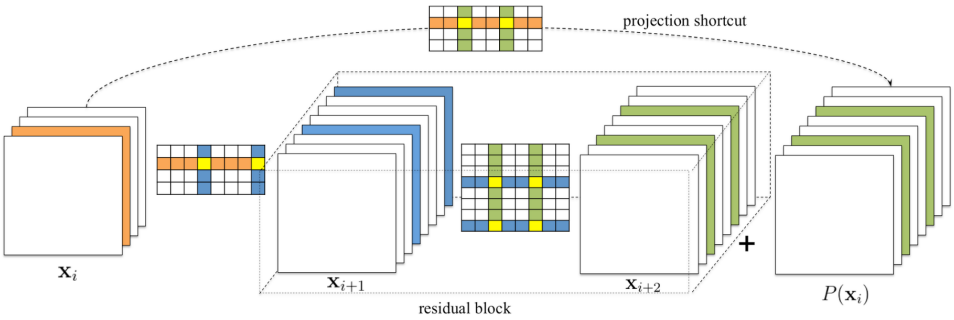


Figure 6: Structured pruning in convolutional ANNs (figure reproduced with permission from Li et al. [107]). Each stack represents the channels of the activation state, while the matrices represent the kernels, which themselves are *nxn* matrices of parameters, for each convolutional layer, with each row being the kernels for each channel in the previous activation and each column being the kernels for each channel in the next activation. Decreasing the incoming channel count in the first convolution is represented in orange. The blue pruning decreases the outgoing channel count of the first convolution and the incoming channel count of the second convolution. This is synonymous with pruning both the incoming and outgoing connections of a neuron in a dense layer. The projection shortcut performs a 1x1 convolution and is pruned accordingly as necessary for dimensionality agreement.

function as structural learning, integrating weight tuning with architectural change to arrive at a final static network in terms of both parameters and architecture. Layer-searching NAS algorithms that use structural learning generally employ a hand-designed super-network that contains all possible layer types over all possible interconnections, then perform the operator of neural pruning after optimizing the super-network to derive the final architecture.

Biological inspiration: In the adult brain, programmed neural cell death is most common following neurogenesis. During early development, neurons must develop sufficient active connections to avoid the default fate of removal [65]. This is similar to one-shot NAS methods such as DARTS [19] where large super-networks are created at initialization with only a minority of neurons remaining in the final architecture; however, biological cell removal is progressive and occurs at different rates in different regions of the brain, more similar to Maile et al. [18]. Neural cell death also follows neurogenesis in the adult brain; in the olfactory bulb, only 50% of newly generated neurons survive to be integrated into existing circuitry, with neural activity being a critical factor in determining cell survival [55]. This is most similar to neuron-level grow-and-prune methods such as NeST [68] and activity-based neural pruning [69, 105, 108], but further research of structural learning methods inspired by adult neurogenesis and subsequent neuron removal is needed.

Architecture Specificity: The structural types present within a network require some consideration for structured pruning, depending on the desired level of structure to prune. Many convolutional structural learning algorithms use the higher level paradigm, shown in Figure 2**(c)**, for pruning [101, 107, 109, 110], as depicted in Figure 6. This allows an algorithm to be applied to convolutional and dense layers, even within the same architecture. Some sub-unit pruning in convolutional layers can still be effective for computational efficiency in contrast to unstructured pruning, such as pruning via striding within kernels [111] and limiting the area of convolution [68]. Beyond each layer's type, the surrounding structure and functionality may also be utilized in more specialized pruning: Kang and Han [69] prunes channels that tend to have low signal after the standard subsequent batch normalization and ReLU operations, while Gordon et al. [112] regularizes and prunes channels via the batch normalization scaling parameters. Exploiting such known structures in the limited architecture space of those including these structures has a trade-off between search space flexibility and effectiveness within those search spaces. When performing structural learning on units, the algorithm must ensure that all intermediate activations maintain compatibility with all incoming and outgoing layers: for example, neurons tied by skip-connections can be grouped during structural operations [112].

Measuring Entities: Structured pruning with heuristic search requires metrics or selection methods that can be measured or enforced at the level of desired unit of pruning. Similarly to unstructured pruning, early methods used perturbation-based selection [99, 113], where the algorithm measures the performance change from removing each neuron individually and retraining the rest of the network, but this does not scale well with the size of the network. One recent exception used a greedy forward search, where all neurons in a layer are pruned and then useful units are individually added back in [114]. For metric-based pruning of units, a norm, across parameters within a unit, of any metrics discussed previously for the unstructured case in Section 3.3.2 can be used [101, 107, 110] or group-sparse regularizers can tie parameters within a unit and encourage sparsity at the unit level [112, 115]. Selection methods that are only possible on the unit-level include activation-based pruning, where units with low or infrequent activations are removed [69, 105], and relative metrics of redundancy, where units with redundant functionality compared to others are marked for removal [109]. One issue with the latter approach is time complexity, since it requires a pairwise comparison of all units within each layer. Unit-wide masks, where the mask controls whether each unit is active or not, are also common, such as in Chen et al. [116], Wan et al. [117], Guo et al. [118] at the convolutional neuron level and in most NAS works at the layer level.

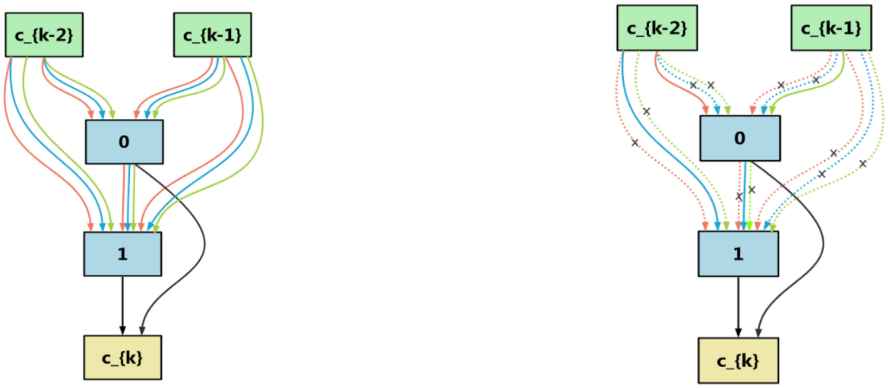
The two main mask-based developmental NAS approaches at the layer level are continuous NAS and path-sampling NAS, as shown in Figure 7. These are used in order to both train the network parameters and evaluate different architecture possibilities, which may be represented as architecture parameters.

In continuous NAS, a continuously structured super-network is trained and then discretized into the desired architecture, as shown in Figure 7a [19, 106, 119–125]. The structural learning, namely neural pruning of the units of layers, occurs at discretization, which may happen once at the end of the search phase or progressively throughout. Before the super-network is fully discretized, it evaluates mixtures of potential layers over potential connections as weighted sums at each activation state, and uses back-propagation of the loss error to strengthen or weaken the architectural weight of each layer. Selecting the architecture from the super-network by the end of the search using learned architecture weights often looks very similar to magnitude-based pruning, selecting the strongest options such that a valid architecture is formed.

For path-sampling NAS, potential paths are discretely sampled from the super-network for training the network's parameters within each path, as shown in Figure 7b [72, 118, 126–129]. During the search process, a discrete path is selected for the training step of each mini-batch, and only the network weights and architecture parameters along that path are updated. Thus, each path is equivalent to performing ephemeral neural pruning for a single forward and backward pass, while persisting and updating weights of the super-network. Path selection may use the architectural parameters as sampling or state-transition probabilities within the super-network [118, 126–128] or use a uniform distribution [72]. At the end of the search process, the highest probability path is selected as the final architecture. While path-sampling NAS allows training passes to occur in a discretized network more structurally similar to the final desired architecture than the continuous mixtures in a super-network, only the parameters on the sampled path, rather than all parameters, have gradient information and thus can be updated per mini-batch.

Scheduling Pruning: As with synaptic pruning, neural pruning has been trending towards iterative methods, even as often as every iteration [130]. In NAS, the earliest continuous NAS methods had a single discretization where the pruning neural operator occurred, after searching the continuous super-network and before evaluating the discretized architecture [19]. However, this can lead to a discretization gap, where the shallower continuous super-network and deeper discretized architecture are too structurally different and have uncorrelated performance, due to the continuous parameters not being close enough to the discretized parameters [81]. More NAS algorithms are incorporating progressive discretization [122, 124], among other techniques [81], to avoid this gap.

As for the training portion of scheduling, NAS methods also tend to do a full reinitialization and retraining of weights after discretization, whereas most other structural learning methods only fine-



(a) Continuous NAS.

(b) Path-sampling NAS.

Figure 7: Example of a repeated cell of a super-network in NAS (figure reproduced with permission from Yao et al. [129]). Within each cell, the blue nodes represent intermediate activations between layers. Each colored edge represents a different layer type, such as a skip-connection or convolutional layer. This cell, c_k, receives the output of the two previous cells, c_{k-2} and c_{k-1}. During network training, in continuous NAS, all layer options are used at all connections, and a weighted sum is computed at each node using the architecture parameters. In path-sampling NAS, a path representing a valid architecture is selected within the super-network, which is a form of ephemeral discretization. A common architecture constraint for cell-based NAS is at most one layer option per connection and exactly two inputs to each node. For both types of NAS, the final goal is to optimally prune the network into a discrete architecture within this constraint like **(b)**.

tune the network after the search and pruning process, as noted in Table 2. As for network compression, Liu et al. [131] finds that complete reinitialization and training is generally superior while Ye et al. [114] finds fine-tuning inherited weights is better, provably for the case of shallow networks and empirically for deeper networks.

Avoiding the pretraining step is more difficult in structured pruning than unstructured, since structural pruning selection techniques often assume all units are trained enough in order to differentiate their function from each other and fairly compare their utility. Pretraining is particularly important for comparing parameterized units with unparameterized units in NAS, like comparing a convolutional layer with a skip-connection at the same location within an architecture. Some relatively zero-cost NAS metrics that measure entire architectures without training, such as measuring saliency or sensitivity to perturbation [132] or nonlinearity alignment at initialization [133], have been shown to be effective in identifying good candidate architectures, which is particularly useful to avoid the expensive search training process when the final network is often retrained from scratch.

Handling Pruned Entities: At the extreme end of ephemeral pruning on the unit level in frequency is Dropout [134], which is a technique of randomly and independently selecting units to momentarily prune for each batch during training, preventing co-adaptation and overfitting. Structural pruning algorithms use ephemeral pruning in a more principled and informed manner, as in path-sampling NAS. This allows the modules of the super-network to learn more independently than in the continuous case. This idea is also used by Yuan et al. [130] at the neuron level, but most other neuron pruning works choose to instead reinitialize new neurons as the creation operation, which will be discussed in Section 3.4.

Impact on Weight Optimization: Structured pruning is less intuitive to consider in the objective function search space, since dimensions are tied through the units containing their corresponding parameters. Thus, multiple dimensions are removed at once for each unit removed. Pruning metrics often gain information through continued training, but this must be balanced with the cost of training units that will be pruned and thus not used in the final network or in further pruning decisions.

Many NAS works pose the architecture search as a bi-level optimization problem, where the archi-

tecture parameters α are optimized given that the weight parameters w are optimal for any given architecture and constrained to certain sparsity rules S that only permit valid architectures [18]:

$$\min_{\alpha} L_{val}(w^*(\alpha), \alpha) \tag{1a}$$

$$\textbf{s.t. } w^*(\alpha) = \arg\min_{w} L_{train}(w, \alpha),\ \alpha \in S. \tag{1b}$$

This problem is intractable to solve directly, so the process is usually approximated by continuous relaxation of α, persisting weights across the architecture optimization search, and alternating weight updates with architecture updates. The gradient updates to the architectures variables may only be locally approximated: using a second-order estimate of the gradient of α that incorporates an inner update of w^* only gives minor improvements in performance for a higher computational cost compared to first order estimates that approximates w^* as the current w [19]. See Liu et al. [135] for a further discussion on bi-level optimization in ANNs.

3.3.4 Pruning Conclusions

In ANNs and in the brain, more information is available for existing neurons and synapses to be pruned versus creating new units. This trend is noted in artificial algorithms, both in the popularity of pruning versus creation as well as more specifically using masking to propagate gradient information to inactive units, thus turning a structural learning problem into a pruning problem.

Pruning in both the brain and ANNs can improve performance relative to cost: the main costs in both medias are time and space, but the realization of these costs change how efficiency is achieved. For example, an important difference between brains and ANNs is the locality of physical biological signals versus the globality of ANN addressing in memory. Biological neurons are limited by physically transported molecular signals, but do not have a scaling time cost for enumerating the local options. ANNs algorithms have no distance-based cost and thus may make decisions globally, but do have a linearly scaling time cost for each entity that is evaluate due to the hardware's bound on floating point operations per second.

Pruning, however, does come with costs, notably the wasted training time of the associated parameters and potential disruption to the learning process. Pruning algorithms must thus strike a balance of information that could be useful for pruning and later utility in the network if the unit is not pruned versus the cost of obtaining this information.

Pruning has been used throughout the history of deep ANNs to partially automate the structural design. However, it relies upon a predefined and instantiated search space. Currently, most methods explicitly hand-design this search space as the initial architecture. Others, as well as the brain, use creation to dynamically create the search space. These creation methods and their synergies with pruning will be discussed in the following section.

3.4 Neuron creation and synaptogenesis

Adding to a search space is naturally more difficult than removing, because an indefinite number of ways to add new elements exists, but removal is limited to only existing elements. For the unit operators of structural learning, this means that neural creation is a much different problem from neural pruning. For the connection operators, because synaptogenesis occurs between existing structures, it is of the same search complexity as synaptic pruning, although metrics for where to add connections are not as straight-forward as where to remove them. Thus, synaptic pruning and synaptogenesis often look very similar, especially for ephemeral structural changes during training.

Nearly parallel to our pruning questions, our main questions for characterizing creation are:

- **Goals of Creation:** Why add connections or units to an ANN?

- **Biological Inspiration:** How can biological neurogenesis and synaptogenesis inspire artificial methods?
- **Architecture Specificity:** How has creation been adapted to different ANN architectures, such as dense layers and convolutional layers?
- **Measuring Entities:** How are the new entities to be added selected over other options?
- **Scheduling Creation:** When and how often does creation occur with respect to optimization of the existing parameters?
- **Operator Interactions:** What is the impact of using pruning along with creation, versus just one such modality?
- **Impact on Weight Optimization:** What is the impact of creation on optimization of parameter weights?

In discussing these questions, we focus on the growth aspects of the papers in our corpus: neural creation and synaptogenesis. We discuss the few topics that span both modalities of creation in Section 3.4.1, then cover each question specifically for synaptogenesis in Section 3.4.2 and for neural creation in Section 3.4.3 before concluding in Section 3.4.4.

3.4.1 Creation Generalities

Not as many shared characteristics exist between the modes of creation compared to those between the modes of pruning. We detail these few general characteristics below.

Goals of Creation: Growing in an ANN allows the architecture to be even more automatically customized: pruning-only algorithms require all entities in the search space to already exist in the initial network. Using the growing neural operators allow a much larger architectural search space to be explored over the course of structural learning, since this search space does not have to be explicitly predefined or instantiated. It is rather implicit from the algorithm's creation operations.

Biological Inspiration: Synaptogenesis is a frequent event in the biological brain which defines the possible neural circuitry using information from genetic cues and neural activity [51]. Neurogenesis, on the other hand, mostly occurs during early development, creating the critical structures of the brain largely through genetic influence [42, 44, 46]. Neural activity regulates both processes but is especially important for synaptogenesis [7, 136]; physical proximity and chemical neuromodulatory signals also influence these processes. As in ANNs, creation is used to expand the possible neural circuitry with relatively uniformed propositions; in both cases, creation is a more difficult problem than removal with less information available.

Impact on Weight Optimization: Creating entities in an ANN expands the dimensionality of the parameter search space. Network morphisms are methods of adding entities to an ANN with a specific weight initialization that preserves the functionality of the network [137]. After using a network morphism to grow units or connections, the expanded ANN may then begin gradient descent at an equivalent location of the previous objective function space but now with expanded dimensions. This guarantees that the new optimal point will have at most the same cost if not lower, since the previous optimum will necessarily be included in the new search space within the subspace where all new parameters are nullified. However, adding too many dimensions increases the time and space costs for the ANN, including in search time if the search space becomes unnecessarily complex. Similarly as for pruning as discussed in Section 3.3.1, creation is generally composed of greedy search decisions in a local search, which can only find local optima.

In the following section on synaptogenesis, we focus on algorithms that have a specific method for growing connections, rather than just reviving previously pruned connections, previously discussed as ephemeral pruning in Section 3.3.2, or creating units with their associated connections, to be discussed in Section 3.4.3.

3.4.2 Growing Connections

Goals of Creation: Synaptogenesis can be used in an ANN to create new connections between existing units. It has been used with pruning for discovering more effective flexible wirings between channels or neurons within ANNs, particularly without an initial overparameterized dense architecture [68, 92–94, 99, 100]. In other applications, Kim et al. [138] augments an ANN's performance after training by adding neuron-level connections directly from hidden neurons to the output, while Elsken et al. [71] searches for layer-level skip connections, among other network morphisms, concurrent with continual training during evolutionary architecture search.

Biological Inspiration: Biological synaptogenesis is an integral part of learning which has been demonstrated in memory and control tasks [89, 139]. While a considerable part of biological synaptogenesis only reinforces the existing connection between neurons through adding new synapses, previously unconnected neurons can also be linked through synaptogenesis. The information used for this process is based on activity and chemical signatures [136]; nearby neurons will exchange signaling molecules to make them more receptive to synaptogenesis [52]. Physical proximity is an important factor in biological synaptogenesis; for example, in a cortical column, two neurons can connect through a short extension, where connection to a neuron to a more distant region requires investing space and energy in growing axon branches with a more daunting problem of connection choice [102]. There are zones of higher connection density, such as cortical columns which have 10^9 synapses for 10^5 neurons, as opposed to the overall higher sparsity of the brain, 10^{15} synapses for 10^{11} neurons. This sparsity means that, through rewiring, a neuron can drastically change its role in information passage [102], and synaptogenesis is the main catalyst of rewiring change. Further work similar to Dai et al. [68], Mocanu et al. [92], Wortsman et al. [94], which connect previously unconnected neurons during learning, warrants exploration, as the brain demonstrates the importance of forming new connections during learning. Neural activity could form the basis for possible connection, as in the brain [140], by detecting neurons or layers in ANNs which have correlated activity and could be directly linked.

Architecture Specificity: Neuron-level synaptogenesis algorithms can generally be applied in both convolutional and standard layers. For convolutional layers, synaptogenesis is generally done on the kernel level [68, 94, 100]. Only Bellec et al. [93] performs synaptogenesis on individual parameters within each kernel, but does so randomly on masked connections to balance informed synaptic pruning. Wortsman et al. [94] introduces a rewiring algorithm that is agnostic to whether connections are convolutional or dense and removes the constraint of layer organization that engineered deep architectures follow.

Measuring Entities: Metrics for synaptogenesis are less straightforward than for synaptic pruning, since the potential connections do not have active information flow. Thus, a common approach is to use a mask that inactivates connections during the forward pass but allows propagation of the gradient to them during the backward pass, providing many of the same metrics as are normally available [68, 94, 100]. To shrink the quadratic search space of all possible connections, Dai et al. [68, 100] only allow connections between neurons in neighboring layers, while Wortsman et al. [94] allows connections across multiple layers but still limited within modules of similar depth. Bellec et al. [93] also uses a mask, but rather performs random synaptogenesis of a currently dormant connection each time another connection is pruned in order to maintain the sparsity level of the network. While masks bound the search space, that is not necessarily limiting to synaptogenesis since it already can only occur between existing units, whether or not a mask is used.

Beyond mask-based approaches, algorithms often use improved performance as the final metric for adding connections. Puma-Villanueva et al. [99] ranks all possible connections with a mutual information heuristic amenable to synaptic creation and adds those that improve the performance. Kim et al. [138] evaluates the effects on performance exhaustively for a smaller search space, only allowing hidden neurons to be connected directly to output neurons. Using improved performance as the selection criteria is intuitive, but is a greedy selection towards the goal of optimizing performance

and is expensive to complete exhaustively without heuristics, limited search spaces, or randomized evaluations.

Scheduling Creation: Synaptogenesis is usually performed iteratively and greedily. New connections need to be optimized after a random initialization for optimal performance, so synaptogenesis phases are generally followed by weight optimization phases of either only the new parameters or the entire network.

Operator Interactions: Most synaptogenesis algorithms also implement synaptic pruning [68, 93, 94, 99, 100]. Implementing synaptogenesis with synaptic pruning can create dynamically balanced connectivity, which may counteract any negative effects of earlier greedy synaptic additions to escape local optima. The remaining synaptogenesis algorithms only use synaptogenesis out of our two connection operators [71, 138]. This simplifies the search process, but may approach local optima due to greediness.

Impact on Weight Optimization: Synaptogenesis adds dimensions to the search space. Because any added synaptic parameters can be set to zero to nullify their function and thus preserve the objective function of the ANN, synaptogenesis can guarantee the new global optimum has at most the same cost. However, depending on the initialization and training algorithms used, the synaptogenesis may displace the current location in the objective function space far enough to descend towards a different, possibly worse or better local optimum. Additionally, adding too many dimensions increases the time and space costs for the ANN, including in search time if the search space becomes unnecessarily complex.

3.4.3 Growing Units

Goals of Creation: Neuron creation, or adding units to an ANN, is the most open-ended neural operator. The general aims of neuron creation are to improve performance and to create an architecture that is specifically effective for the desired task without manual design. The resulting networks should have either reduced space and time complexity for deployment at a desired performance level or a better performance potential. Beyond algorithms with these general aims, some NAS approaches incorporate neural creation in the form of changing the architectural search space over the course of training [72, 119, 125], which provides a larger search space than traditional pruning-only approaches. Both the general and NAS-specific applications allow for a more customized architecture with less hand-design.

Biological Inspiration: Neurogenesis in the biological brain is more constrained than in ANNs; neural progenitor cells must be properly located to create new neurons, there must be enough space, and the associated energy cost is non-negligible. The most common period of neurogenesis, early development, is largely guided by genetic cues [42], with only highly irregular neural activity patterns disrupting neurogenesis [7]. In humans, adult neurogenesis is constrained to a few regions, notably the hippocampus [47, 48], and it is common in other organisms [141]. Adult neurogenesis could serve as further inspiration for structural learning in ANNs as there are fewer constraints. For example, new neurons in the mature brain first develop input synapses which observe incoming connections before firing action potentials [55]; in other words, new neurons are well-initialized based on surrounding activity, similar to recent studies in ANNs [20, 142]. The conditions leading to adult neurogenesis are the subject of current research in neuroscience [143] and could lead to further inspiration in ANNs.

Architecture Specificity: Neuron-level creation usually adds neurons of a predetermined type to existing or new layers. However, since selection techniques are often agnostic to layer type, many neural creation algorithms can be applied to architectures with layers of various types like convolutional or dense [20, 101, 110, 142, 144]. Similarly to pruning, growing new filters in a convolutional layer also involves a mapping in the subsequent layer [101], as shown in Figure 6. Layer-level creation often extends layers with the same type as previous layers [145, 146]. Roberts et al. [125] avoids

pre-engineering the layer type selection by parameterizing the layer function in a search space that includes most commonly used layer types in NAS among many others.

Measuring Entities: Selecting where and how to add new units to a network is not straight-forward. In order to avoid explicitly bounding the search space, a mask cannot be used as it requires all elements to be instantiated to be associated to an element of the mask. Most approaches limit the instantaneous search space of possibilities, such as iteratively adding a single neuron to an existing layer [68, 110, 147–152], adding a single new layer beyond the current architecture [110, 146, 148, 151, 153], or splitting existing neurons [101, 110, 144]. This results in a simpler problem at each step, but stills gives the algorithm the power of an unbounded search space over the course of training.

The simplest approaches, often combined with some form of pruning, use a generic creation scheme that adds some default number of new units [112, 113]. Slightly more informed approaches use the network's objective function as a heuristical measure of success, adding the units that reduce the error the most out of generated options to the network [68, 153]. Evci et al. [142] adds neurons that maximize the immediate improvement in performance, while Maile et al. [20] dynamically adds neurons with unique activations or weights to avoid redundancy. Liang et al. [154] adds special exponentially activated neurons that provably improve the loss landscape.

Scheduling Creation: The objective function is used not only for creation selection but also for iterative scheduling and termination. In early construction algorithms, often a single neuron was added at a time upon convergence of the current network until the desired performance is reached [99, 147–152]. This approach of a single neuron added between training to convergence does not scale very well with the size of the network or complexity of the task. Many of the recent methods still use performance stagnation during training as a trigger for a neural creation phase, but for adding either many new neurons or an entire layer at a time [71, 144, 153]. Otherwise, a manual schedule for iterative phases may be implemented [101, 113, 118]. Most of these schedules introduce hyperparameters, such as thresholds or durations per each phase of the schedule. These require hand-tuning or optimization for each specific dataset, task, and initial architecture. This is not desirable for generality and automation of algorithms across applications, but selecting hyperparameters is a much smaller search space for manual exploration than that of the architecture being automatically designed. Maile et al. [20] uses a dynamic schedule that adds neurons while novel directions to explore in either the activations or weights in that layer still exist.

Operator Interactions: Most of the recent neural creation algorithms also use neural pruning, while the older ones did not. This dynamic approach not only can ameliorate effects of greedy structural changes, but also leaves the size of the final ANN as either an objective to optimize in tandem to performance or as an open-ended result. For example, Guo et al. [118] can adapt initial seed networks to any computational budget, whether smaller, approximately the same, or larger than the original network. Connection operators may also be used in addition to unit operators to make finer adjustments to the network [68, 99, 101].

Impact on Weight Optimization: Neural creation expands the dimensionality of the search space of gradient descent. To determine the starting point in the newly expanded space, the weights of new units need to be initialized before they are trained. Unit-splitting algorithms [101, 110, 144, 145, 155] copy the weights and may add a small perturbation to ensure the units are not redundant if split in parallel. Network morphisms, named by [137] and used in many other works [20, 71, 119, 142, 146, 152], are structural operations that initialize new structures to be null, which generally means initializing new weights to be zero. Most other neural creation algorithms use a similar initialization scheme to the original network, such as random weight initialization. Both unit-splitting without random noise and network morphisms perturb the location of the network in the new space less than methods that use random noise in their initialization. Both cases can be beneficial: staying in a similar location with newly added dimensions allows the path of gradient descent to continue with less disturbance or risk of regressing from the current level of performance, while a small perturbation may help escape local

optima or saddle points [110, 144], as demonstrated in Figure 5.

Liang et al. [154] theoretically shows that the addition of special exponentially-activated neurons to a basic ANN can make all minima globally optimal. However, no empirical work has shown whether these simple additions affect other aspects of training dynamics and performance. For example, additional saddle points may confound the benefits of no bad local minima [156].

3.4.4 Creation Conclusions

Creation has been less well-studied than pruning in ANNs, due to the additional complexity imposed by the open-ended search space of creation, which is less cooperative with back-propagation than static architectures and even pruning. Hand-designing neural architectures has also made it less practical and imperative, but the desire for it is continuing to increase alongside computational capabilities.

The brain has rather distinct phases of early development and adult life. These correspond roughly to defining the neural architecture search space, or hand-design of an ANN architecture for the case of conventional non-structural learning, and searching within this space over the process of learning. Most structural learning research focuses on automating and improving the latter process, leaving further optimization of search space design as an open problem. However, the search space design is also synonymously influenced by the genetic priors used in the brain that have been optimized over the course of human evolution, making search space design a much broader problem with a larger scope than one-shot learning. The creation operations allow for a more dynamic approach to search space design.

The brain uses Hebbian learning, or reinforcing effective connection strengths in response to correlations in activity between interacting neurons, as one of many mechanisms for dynamic structural learning [57]. Synaptic scaling, or the post-synaptic homeostatic plasticity [157], and non-synaptic plasticity, such as membrane conductance changes [50], counterbalance this strengthening and are analogous to weight decay and batch normalization, respectively, in ANNs. The explicit use of Hebbian learning in ANNs, which could be implemented as increasing weight magnitude between artificial neurons with correlated activations, is thus far not as popular as back-propagation but has been done [158]. Standard ANNs are not time-dependent, which changes the nature of correlations in activity compared to that of the brain, and ANNs have global information access, permitting back-propagation of errors more easily than in the brain, although there are theories of supervised learning via target activity patterns [159]. Dai et al. [68] applies this technique to the creation operations and shows that growing connections via larger values of the gradient of the network loss with respect to dormant masked weights is a Hebbian-like rule. This suggests further exploration into appropriate application for artificial media of Hebbian and other plasticity theories such as behavioral timescale synaptic plasticity [160] and .

As previously discussed in Section 3.3.4, the physical brain excels at local communication without indexing cost while ANNs can operate globally but with a cost per index. This has implications for effective creation in ANNs: to maintain search efficiency, non-random creation in ANNs, where each potential entity must be indexed and evaluated, should be strategically bounded at a given structural step without overly restricting the size of the overall search space. Creation is not as explicitly bounded in brains: unit creation, including migration, is guided by genetic and developmental factors while synaptogenesis is guided locally by chemical signals, yet there are no restrictions such as having predefined patterns of connections for each neuron. For synapse measurement, the brain does not use simulated heuristics like ANNs often do, but instead selects connections by dynamic processes of repeated synaptogenesis and synaptic removal as directed by various types of plasticity [50, 57, 157]. Similarly for neurons, the heightened neurogenesis of early development is balanced by the automatic removal of new neurons unless they prove their utility. This suggests an approach for ANNs to make sub-optimal but less expensive additions balanced with more careful pruning to dynamically and efficiently construct an informed architecture.

3.5 Perspectives

With our deepened understanding of how the four neural operators work in ANNs, we present our perspectives on the state of structural learning. We first note implementational trends that shape current works in Section 3.5.1. We then discuss current challenges with future directions in Section 3.5.3.

3.5.1 Implementations and Trends

In order to successfully implement structural learning on modern machines rather than biological media, we note several approaches used commonly across algorithms. The ongoing development of computer hardware and software, which provide the media on which ANNs are implemented, has potentiated each of the innovations in ANNs throughout history. Present computational technology is particularly adept at handling array-based calculations. For example, GPUs drastically speed up ANN training wallclock time particularly with stochastic gradient descent (SGD) and tensor-based architecture parameterizations and data structures. However, they also impose a limit in random-access memory (RAM): ANN training is much more efficient and easier to implement if the entire memory-intensive backward pass of SGD can fit on the GPU's RAM at once. The inference computational cost of an ANN is often measured in the number of Multiply-Adds, or equivalently floating point operations (FLOPs), required per inference of a single input. This value can be used as part of a multi-objective approach, maximizing performance while minimizing the inference cost. The training cost is usually measured in GPU-days, the number of days to train on a single GPU using current hardware and software. Low training costs are a key benefit of structural learning over techniques that instantiate and train a statically structured model for many individual architectures. The following implementation trends are noted across our structural learning corpus, specified in Table 2.

Modularity: Dividing a design problem into multiple hierarchical levels and reusing modules learned at lower levels makes it more tractable. This is seen biologically, where symmetry and modularity can be seen from the genetic level, reusing sections of DNA for many proteins that can each have many functions, through the nervous system level, showing bilateral and other symmetries, and even further, as in convergent evolution where unrelated organisms evolve similar structures or functionalities.. For ANN architectures, this entails searching for smaller modular structures that can be used as building blocks to build larger architectures. This technique has already been used in the hand-design of architectures, like ResNet [29]. It is also used in structural learning, notably unit creation and pruning where groups of parameters are added or removed as a unit. Modularity has especially been used in NAS in order to reduce the complexity of the search space. Most NAS algorithms that allow flexible interconnectivity use repeated cells of the same architecture during the search process. Some recent NAS approaches incorporate structural learning on multiple levels, such as searching for both layer-level and neuron-level structures [106]. NAS algorithms incorporating creation take this even further by also optimizing the layer options used [72, 119, 125], thus expanding the search space. Modularity is particularly capitalized in algorithms that either duplicate or split units [101, 110, 144, 145, 155]. These algorithms reuse neurons, layers, or branches of the network that have already been trained on the present dataset and task as initialization points.

Masking and Super-networks: A common approach to allow structural learning is to use masking within the ANN implementation. This mask is an encoding of the structure, enumerating all possible entities and controlling which ones are actively used in a given structure. By relaxing this mask to be continuous, the mask can permit continuous learning where the mask itself is treated like parameters of the network that can be optimized and discretized, such as in pruning or continuous neural architecture search.

When implemented on the layer level for NAS, masking also naturally leads to weight-sharing, where many different architectures are represented with shared parameter data structures but the mask imposes different functionality by specifying the architectural path and thus the parameters used within the super-network. The architectures parameters used for this specification in continuous

NAS are usually activated by softmax, gumbel-softmax, or another balancing function in order to enforce scaling across parallel options. Gumbel-softmax and annealing are more recent innovations in activation of NAS parameters to smooth the transition to the final discretization [69, 117, 123, 128]. Other NAS papers have introduced non-competitive activation functions [161]. Once activated, the architecture parameters may then be used for weighted sums of the activations from parallel layer options in continuous architecture search. For path-sampling architecture search, the activated parameters are often used as a sampling probability.

When implemented on the neuron level, masking for synaptic pruning, such as in unstructured pruning algorithms, allows for simpler computations by element-wise multiplication, but does not lead to any computational speed-up without specialized software. Because masks for neural pruning control multiple synaptic parameters per unit with a single masking parameter, they may require regularization to enforce group-wise operations on the parameters during the algorithm, or else the mask may be implemented across dimensions of the parameter matrices if that is compatible with the parameter data structure representation, such as batch normalization scaling factors [112]. Like NAS, some neuron-level masking implementations also use temperature in their implementation to smooth the transition from the continuous relaxation to discrete masking [93].

Masking naturally turns architectural selection problems into a pruning problem with a bounded search space. This often creates a limitation in neural architecture search and pruning methods where these methods are used: the maximum size network over the course of search should be able to fit its forward and backward passes on a GPU at the same time, even if the desired final network will be much smaller. For example, super-networks in NAS are often limited by the GPU size: for continuous NAS methods, all options are at least partially activated, resulting in a very large network to evaluate and optimize. In order for the final network discretized from the super-network to also approach the GPU size, many continuous NAS works search in a shallower architecture with fewer channels and then implement the discretized structure in a deeper architecture by repeating cells, which is another benefit of using modular cells. However, this restrains the architecture search to repeatable cells with channels, which is not always applicable for tasks outside of image classification. It also further deepens the discretization gap by contributing to the structural differences between the super-network and the discretized network, as discussed in Section 3.3.3.

Dynamically sparse training methods avoid the large initial model by sparsely initializing the network and dynamically adjusting which entities are active, allowing the network to maintain a maximum density level [92–94]. This results in a predefined structural search space without limitations on the size of the full search space, but requires structural learning processes that are not memory-bound to fully reap the benefits over standard pruning techniques.

Interconnectivity: The brain shows connectivity patterns that are much more flexible and complex than modern standard feed-forward networks, where neurons are organized in layers and each layer is only connected to its immediate neighbors. Neural architectures tended towards this organized structure in the early decades of backpropagation, particularly for the multi-layer perceptron. Recently, the skip-connection broke this simple patterning and improved the state of the art in computer vision and many other tasks [162]. It allows for much deeper networks to be trained without vanishing gradients and for higher and lower-level features to be used together. Many structural learning algorithms like NAS allow for interconnectivity flexibility on the layer level, while Bellec et al. [93], Wortsman et al. [94] allow even finer interconnectivity outside of layer organization on the neuron level. Including the potential for such flexible interconnectivity in structural learning, rather than using a rigid connectivity backbone, greatly expands the search space and thus adds complexity to the search, but also yields the potential for more powerful networks.

Weight Reinitialization: Most neural-level structural learning algorithms are end-to-end, where weights learned in tandem to the architecture are used in the final ANN either without any further training or with only fine-tuning. In contrast, most layer-level continuous NAS approaches require full weight reinitialization and training after architecture discretization. While this does allow the

super-network and discretized architecture to have different shapes and training hyperparameters, it is generally more expensive and has a risk of uncorrelated performance between the two stages. This trend is also seen in Frankle and Carbin [12], although here the pruned network returns to the initial initialization for the remaining weights.

3.5.2 Adjacent Domains

In addition to the external review papers cited in Section 3.2.2, we build upon our analysis to mention closely related domains and techniques that are not quite covered by our structural learning framework. They generally cover neural timescales or non-structural operations outside of the scope of our neural operators, but could be used in tandem with or inspire other forms of structural learning.

Attention and LSTMs: In contrast to the methods we have considered that specifically use dynamic connectivity and structures during the training process, networks incorporating self-gating mechanisms, such as recurrence and attention, exhibit ephemerality at inference time [75, 76]. This is most analogous to computation on the neuromodulatory timescale. Attention operates similarly to regulatory firing, instantaneously adjusting computation for a single input without longer term effects using feedback connections [163]. Simpler recurrence structures can also operate on this very short timescale, while LSTMs are more akin to chemical neuromodulation [164].

Biologically-inspired Networks: There are a number of artificial neural model and network types which are further inspired by biology. Spiking neural networks (SNNs) use the transmission of discrete spikes, sometimes along multiple synapses between a given pair of neurons, to transmit information [165, 166], thus allowing for spike-timing dependent plasticity and stochasticity [167]. Liquid state machines, a recurrent version of SNNs, have been studied using similar neural operatorsas the ones studied in this work [168]. As SNNs imitate the spiking behavior of biological cells, they can be used to model biological networks, including network structure [169]. Simple organisms such as *C. Elegans* with mapped neural circuits can be modelled with high levels of detail [170]; neural circuit policy models, inspired by the brain structure of *C. Elegans*, use cellular dynamics that model biological neurons, including time dependence and sparsity [171]. Modelling biological neural structure can help understand how such structures form and their role in learning, and biologically-inspired networks are well-positioned for this.

Evolutionary Algorithms: Evolutionary algorithms are a logical choice for NAS due to their flexible problem encoding and high parallelization; in addition, searching for the evolutionary priors for effective learning is a clear inspiration from biology. There is a long history of using evolution to find ANN structure: Miller et al. [172] was among the first to describe architecture search using a Genetic Algorithm; in Richards et al. [173], ANNs were evolved to play Go; and NEAT [74] has been applied in domains from image generation [174] to Atari game playing [175]. Contemporary methods combine evolutionary architectural search with gradient descent on weights [72, 73]. In these methods, structural changes usually happen at the generational level, for example as mutations of existing architectures; while this makes them difficult to study in the context of individual neural operators, they can provide insight and comparison for structural learning methods, as highlighted in Stanley et al. [82].

A different evolutionary domain which takes direct inspiration from the biological evolution of nervous systems is the evolution of structural learning rules [83, 176, 177]. In these works, structural learning decisions such as performing neural operators are taken by evolved rule sets, in the form of L-systems [178], grammars [176], or functional graphs [177]. In these works, individuals in a population representing different structural learning rules are used to develop ANNs; structural characteristics of the resulting ANNs or the performance of the ANN on tasks are used to inform the fitness and optimization of the various rules. Miller [83] presents a comprehensive overview of this domain. As demonstrated in this chapter, structural learning decisions can be difficult to design, with aspects like timing, relation to learning, and information from other parts of the network contributing to the

decision. Evolution can aid in this design, and we aim in this chapter to provide information about structural learning which may help inform these evolutionary approaches.

Meta-learning: The fields of AutoML and more specifically meta-learning go beyond optimizing a single model and rather tackle optimization of how a model learns [31, 179]. Many meta-learning works aim for fast learning, using only a few samples to learn a new task. Humans and other animals developed fast learning, such as easily learning how to toss a new object after learning how to toss other objects, over the course of evolution; we cannot nor need to conscientiously change brain structure to learn a new skill, while genes and structural operations in the brain are separated by many processes that prevent any strong direct effects. Thus, while fast learning itself happens on a timescale between neuromodulation and structural learning, the technique of fast learning was discovered over the course of the much longer timescale of evolution.

Similar hierarchies of automation occur in meta-learning and large language models, even going towards networks building networks [180, 181]. In these approaches and evolutionary search, there must be hand-design at some level, but engineering at a higher level allows for higher complexity to occur throughout the subsequent lower levels. In the biological case, operations within neural networks are abstracted away from the two controllable variables of consciousness and genes. This suggests continuing setting up artificial systems from progressively higher levels, but also comes with the disadvantage of turning the abstraction between levels into a black box. Similarly, meta-learning works often by training ANNs in the pursuit of learning how to improve learning [181, 182], although the recent trend of massive models with attention mechanisms allowing sparse module activations trained on massive amounts of data and tasks exhibit fast learning as well [183].

With the current state of structural learning and adjacent domains in mind, we next present challenges and future directions within structural learning.

3.5.3 Current Challenges and Future Directions for Development

As computational speed and power, as well as datasets, continue to increase, the desire to automate the engineering of ANN architectures is increasing as well. Human engineering is currently a rate-limiting step of ANN architecture innovation. Among discussions already presented throughout this work on the past and current state of structural learning, we will end our discussion with present challenges and future directions for development.

Many of the benchmark applications that structural learning algorithms are implemented and demonstrated for are well-studied supervised tasks over datasets with static and representative distributions across the training and test partitions. While these provide a consistent benchmark against hand-designed networks and are useful for algorithm development, their utilization is almost paradoxical for one of the powerful potentials of structural learning: automating structural design even for tasks that are not well-studied. Structural learning may allow for more power in less studied and engineered applications, bypassing the need for specific architectural innovations to be designed by hand for every such task. This phenomenon is particularly noted in NAS; automating the engineering of architectures could potentiate the application of ANNs to less-studied tasks, but the majority of NAS papers only report results on the same few image classification datasets for which architectures have already been thoroughly researched, each trial trained independently from scratch, with image-specific search spaces like convolutional layers in repeatable cells. Learning environments that go beyond the basics include multi-task learning [91, 100, 128, 155], continual learning [91, 100, 110, 120, 184], and reinforcement learning [185–187]. These may also approach the dynamic learning environment of the human brain more closely, where representations, labels, and rewards must be internally implemented [188, 189]. Few structural learning papers have implemented their algorithms in these environments thus far, but they represent a deeper level of artificial intelligence.

Our definition of structural learning implies dimensionality changes to the objective function space of ANNs. Structural optimization alongside weight optimization thus prompts a multi-level optimization,

with the structure of the network being the outer-most level that defines the space of parameters being optimized on a lower level [81]. This is explicitly considered in many continuous super-network NAS papers as a bilevel problem [19], but the optimization is often simplified to alternating gradient descent steps without theoretical guarantees but with reasonable performance relative to time and computation cost in practice. Multi-level optimization methods are continuously being developed [190] and may be applied to structural learning, but the trade-off of performance with complexity cost must be considered for effective use.

The level of abstraction, demonstrated in Figure 2, has resulting trade-offs for the structural learning algorithm employing it. Lower level abstractions, such as the direct paradigm of feed-forward neurons, allow for fine-grained tuning of the architecture, but may be too minute for architectures with multiple higher orders of structure and billions of parameters. Higher level paradigms, towards NAS super-networks, impose difficulty by requiring additional structural design in the search space, such as selecting or even structurally optimizing the layer options present in each connection as modules. Combining multiple levels within a single algorithm has thus far been rare [106], but presents a powerful direction for structural learning, requiring coordination of each level that have mostly been studied in isolation thus far.

Neural creation is the most difficult form of structural learning because it is open-ended and slow with current methods. However, such a large search space gives it the potential to be the most fruitful in the pursuit of automating architectural design of ANNs. It is represented not only in selecting where and how to add new units, but also in widening the search spaces of potential layer types and connectivity patterns, which are currently rather limited to discrete, hand-engineered choices. In order to surpass this limited space, Bronstein et al. [191] presents a geometric framework to unite diverse ANN architectures, layer types, and data structure types. Applying these ideas to ANN structural learning could expand the search space by utilizing symmetry-inducing invariances with less restrictions on the input data structure, thus supporting applications even to less naturally tensor-like data structures.

In biology, the genetic code potentiates each neural operator within biological media through transcription then translation to proteins underlying the cellular processes that govern brain structure dynamics. In ANNs, the components of the learning algorithm can be considered like the genetic sequence which determines how ANN learning, both basic weight optimization and the more difficult structural learning, can operate within the computational media of hardware and software given an individual in a learning scenario with training data. The developmental search space is encoded in the algorithm. Most such genetic sequences are hand-designed, but some efforts have been made towards recipe search, where the algorithm itself is automatically designed and optimized. Evolution was the crucial process for discovering the current foundation for biological structural learning, such as discovering useful modular structures and defining the phases of high neurogenesis in early development to dynamic connectivity later in life. Some efforts towards replicating this process are in evolutionary recipe search, where an evolutionary algorithm tries to optimize the entire procedure of structuring and training a neural network. This represents learning on multiple timescales, beyond just the lifetime of a single ANN.

Beyond providing a biologically-inspired framework for structural learning in ANNs, we also intend for this work to strengthen connections between the various sub-communities performing structural learning. We noted a lack of cross-citations both across time and across sub-communities. Many pruning algorithms in the last five years implement the same main abstract methodology as those from three decades ago, but now show drastic improvements in performance [192]. While pruning and NAS works are somewhat well-connected within their respective corpora, developmental works using the creation operations are not, due to both a lack of common vocabulary and a relatively low frequency of new methods. Further, algorithms for more dynamically structured ANNs often compare against their more static counterparts, but not vice versa, and a similar pattern occurs for some of the adjacent domains mentioned in Section 3.5.2 versus conventional networks. Our provided framework can help

bridge the abstract similarities between algorithms to promote collaboration, not only between the neuroscience and machine learning communities, but also between the sub-communities of the latter.

In conclusion, structural learning in ANNs is a dynamic and diverse field with a vast potential. Our biologically-inspired framework of neural operators, consisting of neural creation, synaptogenesis, neural pruning, and synaptic pruning, provides a means for synergy not only between neuroscience and machine learning, but also between subcommunities within structural machine learning such as pruning, AutoML, neural architecture search, and developmental neural networks. Gaps thus far identified in the current foundational understanding of structural learning for ANNs are tackled in the following chapters.

	Operator type					Unit type				Layer type			Task						
Reference	*Neural creation*	*Synaptogenesis*	*Neural pruning*	*Synaptic pruning*		*Neuron*	*Kernel shape*	*Filter/channel*	*Layer(s)*	*Dense*	*Conv/pooling*	*Recurrent*	*Multi-task*	*Continual*	*Image classification*	*Other computer vision*	*NLP/sequence*	*Supervised learning*	
Ash [147]	+					+				+								+	
Fahlman and Lebiere [148]	+					+			+	+								+	
Frean [149]	+					+				+								+	
LeCun et al. [38]				+		+				+					+			+	
Hassibi et al. [95]				+		+				+								+	
Lehtokangas [150]	+					+				+								+	
Ma and Khorasani [151]	+					+			+	+								+	
Narasimha et al. [113]	+		+			+				+								+	
Islam et al. [152]	+					+			+	+								+	
Puma-Villanueva et al. [99]	+	+	+	+		+				+								+	
Han et al. [86]				+		+				+	+				+			+	
Guo et al. [97]				+		+		+		+	+				+			+	
Pan et al. [115]			+	+		+				+								+	
Siegel et al. [109]			+			+				+	+				+			+	
Anwar et al. [111]			+	+			+	+			+				+			+	
Cortes et al. [153]	+		+						+	+								+	
Elsken et al. [71]	+	+					+	+	+		+				+			+	
Gordon et al. [112]	+		+			+		+		+	+				+			+	
Li et al. [107]			+					+			+				+			+	
Lu et al. [155]	+								+	+	+		+		+			+	
Bellec et al. [93]		+		+		+					+	+			+		+	+	
Cai et al. [126]			+						+		+				+			+	
Chen et al. [116]			+					+	+		+				+			+	
Liu et al. [19]			+						+		+	+			+		+	+	
Mocanu et al. [92]		+		+		+					+	+			+			+	
Veniat and Denoyer [127]			+					+	+		+				+	+		+	
Bian et al. [193]			+						+	+	+				+			+	
Dai et al. [100]		+		+		+		+		+	+	+	+	+	+		+	+	
Dai et al. [68]	+	+	+	+		+		+			+				+			+	
Du et al. [101]	+		+	+		+		+		+	+				+			+	
Laube and Zell [119]	+		+				+	+	+		+				+			+	
Lee et al. [98]				+		+		+		+	+	+			+		+	+	
Li et al. [120]			+						+	+	+			+	+			+	
Liu et al. [144]	+					+		+		+	+				+		+	+	
Mei et al. [121]			+				+	+	+		+				+			+	
Wortsman et al. [94]		+		+				+			+				+			+	
Yan et al. [106]			+	+		+			+		+				+			+	
Bi et al. [122]			+						+		+				+			+	
Ci et al. [72]			+				+		+		+				+			+	
Dong et al. [145]	+								+		+				+			+	
Guo et al. [128]			+	+					+	+	+		+		+	+		+	
Kang and Han [69]			+					+			+							+	
Noy et al. [123]			+						+		+				+			+	
Wan et al. [117]			+				+	+			+				+			+	
Wen et al. [146]	+								+		+				+			+	
Wu et al. [110]	+		+			+		+	+	+	+			+	+			+	
Yao et al. [129]			+						+		+	+			+		+	+	
Ye et al. [114]			+			+		+		+	+				+			+	
Guo et al. [118]	+		+					+	+		+				+			+	
Kim et al. [138]		+				+				+	+				+			+	
Peng et al. [91]				+		+				+	+		+	+	+	+		+	
Roberts et al. [125]	+		+				+	+	+		+	+			+		+	+	
Sinha and Chen [194]			+						+		+				+			+	
Wang et al. [124]				+					+		+				+			+	
Yuan et al. [130]			+					+			+	+			+	+	+	+	

Table 1: Operator type, unit type, layer type, and task qualities across our corpus.

	Method										Implementation			
Reference	*Mask descent*	*RL generator*	*Evolutionary*	*Magnitude-based*	*1st order metric*	*2nd order metric*	*Other metric-based*	*Improvement-based*	*Regularization*	*Splitting/joining*	*Modularity*	*Weight-share/Mask*	*Interconnectivity*	*Final reinitialization*
Ash [147]							+							
Fahlman and Lebiere [148]							+							
Frean [149]							+							
LeCun et al. [38]						+								
Hassibi et al. [95]						+								
Lehtokangas [150]							+							
Ma and Khorasani [151]							+							
Narasimha et al. [113]							+	+						
Islam et al. [152]							+							
Puma-Villanueva et al. [99]							+						+	
Han et al. [86]				+					+			+		
Guo et al. [97]				+								+		
Pan et al. [115]				+					+					
Siegel et al. [109]										+				
Anwar et al. [111]			+									+		
Cortes et al. [153]			+										+	
Elsken et al. [71]			+									+	+	
Gordon et al. [112]									+					
Li et al. [107]				+										
Lu et al. [155]									+	+		+		
Bellec et al. [93]	+			+					+					
Cai et al. [126]	+	+		+							+	+		+
Chen et al. [116]			+									+		
Liu et al. [19]	+			+							+	+	+	+
Mocanu et al. [92]				+										
Veniat and Denoyer [127]	+				+				+			+	+	+
Bian et al. [193]	+						+	+					+	
Dai et al. [100]				+	+							+		
Dai et al. [68]				+	+									
Du et al. [101]					+					+				
Laube and Zell [119]	+				+					+	+	+	+	+
Lee et al. [98]					+									
Li et al. [120]	+			+								+		
Liu et al. [144]							+			+				
Mei et al. [121]	+				+						+	+		
Wortsman et al. [94]	+			+								+	+	
Yan et al. [106]	+			+							+	+	+	
Bi et al. [122]	+			+							+	+	+	+
Ci et al. [72]	+		+	+								+	+	+
Dong et al. [145]							+							
Guo et al. [128]					+							+		+
Kang and Han [69]	+				+				+			+		
Noy et al. [123]	+			+							+	+	+	+
Wan et al. [117]	+			+								+		+
Wen et al. [146]								+						
Wu et al. [110]					+					+				
Yao et al. [129]	+			+							+	+	+	+
Ye et al. [114]								+				+		
Guo et al. [118]	+			+								+		+
Kim et al. [138]								+					+	
Peng et al. [91]					+							+		
Roberts et al. [125]	+			+							+	+		
Sinha and Chen [194]			+									+	+	+
Wang et al. [124]	+							+				+		+
Yuan et al. [130]	+			+								+		

Table 2: Method and implementation qualities across our corpus.

4 Equivariance-aware Architectural Optimization

Constraining neural networks to be equivariant to symmetry groups present in the data can improve their task performance, efficiency, and generalization capabilities [191], as shown by translation-equivariant convolutional neural networks [8, 195] for image-based tasks [9]. Seminal works have developed general theories and architectures for equivariance in neural networks, providing a blueprint for equivariant operations on complex structured data [196–199]. However, these works design model constraints based on an explicit equivariance property. Furthermore, their architectural assumption of full equivariance in every layer may be overly constraining; e.g., in handwritten digit recognition, full equivariance to 180° rotation may lead to misclassifying samples of “6" and “9". Weiler and Cesa [200] found that local equivariance from a final subgroup convolutional layer improves performance over full equivariance. If appropriate equivariance constraints are instead learned, the benefits of equivariance could extend to applications where the data may have unknown or imperfect symmetries.

Learning approximate equivariance has been recently approached through novel layer operations [201–205]. Separately, the field of neural architecture search (NAS) aims to optimize full neural network architectures [19, 71, 180, 206, 207]. Existing NAS methods have not yet been developed for explicitly optimizing equivariance, although partial or soft equivariant approaches like those proposed by Romero and Lohit [208] and van der Ouderaa et al. [209] do allow for custom equivariant architectures. An important aspect of NAS is network morphisms: function-preserving architectural changes [137] which can be used during training to change the loss landscape and gradient descent trajectory while immediately maintaining the current functionality and loss value [16]. Developing tools for searching over a space of architectural representations of equivariance would allow for existing NAS algorithms to be applied towards architectural optimization of equivariance.

Contributions First, we present two mechanisms towards equivariance-aware architectural optimization. The *equivariance relaxation morphism* for group convolutional layers partially expands the representation and parameters of the layer to enable less constrained learning with a prior on symmetry. The $[G]$*-mixed equivariant layer* parameterizes a layer as a weighted sum of layers equivariant to different groups, permitting the learning of architectural weighting parameters.

Second, we implement these concepts within two algorithms for architectural optimization of partially-equivariant networks. Evolutionary Equivariance-Aware NAS (EquiNAS$_E$) utilizes the equivariance relaxation morphism in a greedy evolutionary algorithm, dynamically relaxing constraints throughout the training process. Differentiable Equivariance-Aware NAS (EquiNAS$_D$) implements $[G]$-mixed equivariant layers throughout a network to learn the appropriate approximate equivariance of each layer, in addition to their optimized weights, during training.

Finally, we analyze the proposed mechanisms via their respective NAS approaches in multiple image classification tasks, investigating how the dynamically learned approximate equivariance affects training and performance over baseline models and other approaches.

4.1 Related Works

Approximate equivariance Although no other works on approximate equivariance explicitly study architectural optimization, some approaches are architectural in nature. We compare our contributions with the most conceptually similar works to our knowledge.

The main contributions of Basu et al. [205] and Agrawal and Ostrowski [210] are similar to our proposed equivariant relaxation morphism. Basu et al. [205] also utilizes subgroup decomposition but instead

algorithmically builds up equivariances from smaller groups, while our work focuses on relaxing existing constraints. Agrawal and Ostrowski [210] presents theoretical contributions towards network morphisms for group-invariant shallow neural networks: in comparison, our work focuses on deep group convolutional architectures and implements the morphism in a NAS algorithm.

The main contributions of Wang et al. [201] and Finzi et al. [202] are similar to our proposed $[G]$-mixed equivariant layer. Wang et al. [201] also uses a weighted sum of kernels, but uses the same group for each kernel and defines the weights over the domain of group elements. Finzi et al. [202] uses an equivariant layer in parallel to a linear layer with weighted regularization, thus only using two layers in parallel and weighting them through regularization rather than parameterization.

In more diverse approaches, Zhou et al. [203] and Yeh et al. [204] represent symmetry-inducing weight sharing through learnable matrices. Romero and Lohit [208] and van der Ouderaa et al. [209] learn partial or soft equivariances for each layer.

Neural architecture search Neural architecture search (NAS) aims to optimize both the architecture and its parameters for a given task. Liu et al. [19] approaches this difficult bi-level optimization by creating a large super-network containing all possible elements and continuously relaxing the discrete architectural parameters to enable search by gradient descent. Other NAS approaches include evolutionary algorithms [71, 206, 207] and reinforcement learning [180], which search over discretely represented architectures.

4.2 Background

4.2.1 Symmetries and Group Theory

A symmetry of an object is a mapping of the object onto itself such that structure is preserved. A *symmetry group* G is a set of such mappings along with a binary operation $\cdot\colon G \times G \to G$, known as the *group product*, that satisfies axioms for closure, associativity, the identity, and the inverse [211]. A group G acts on a set $\mathscr{X}$ via the *group action* $.\colon G \times \mathscr{X} \to \mathscr{X}$, $(g,x) \mapsto g.x$ that satisfies axioms for identity and compatibility: $\mathscr{X}$ is called a *G-space*.

Equivariance is the property of a mapping such that transformation of the input results in equivalent transformation of the output. Formally, a mapping $h\colon \mathscr{X} \to \mathscr{Z}$ between two G-spaces is G-equivariant if for all $g \in G$ and $x \in \mathscr{X}$ we have: $h(g.x) = g.h(x)$. For example, an image segmentation neural network should be $T(2)$-equivariant: shifting the input should result in the same shift in the output.

Invariance is a special case of equivariance, where the output of the function is completely independent of transformation of the input. Formally, a mapping $h\colon \mathscr{X} \to \mathscr{Z}$ is G-invariant if for all $g \in G$ and $x \in \mathscr{X}$ we have: $h(g.x) = h(x)$. For example, an image classification network should be $T(2)$-invariant: shifting the input should not change the output. Symmetries leave objects invariant.

For two groups G and H with group products $\cdot_G$ and $\cdot_H$ respectively where H acts on G with group action $.$, the (outer) semi-direct product $G \rtimes H$ of H acting on G is a group composed of the set of elements $G \times H$ with group product $(g,h) \cdot (g',h') = (g \cdot_G (h.g'), h \cdot_H h')$ and inverse $(g,h)^{-1} = (h^{-1}.g^{-1}, h^{-1})$.

A *subgroup* H of G is a nonempty subset with the same group product that also fulfills the group axioms. Then, $gH = \{g \cdot h | h \in H\}$ and $Hg = \{h \cdot g | h \in H\}$ denote the *left coset* and *right coset*, respectively, of H with representative g.

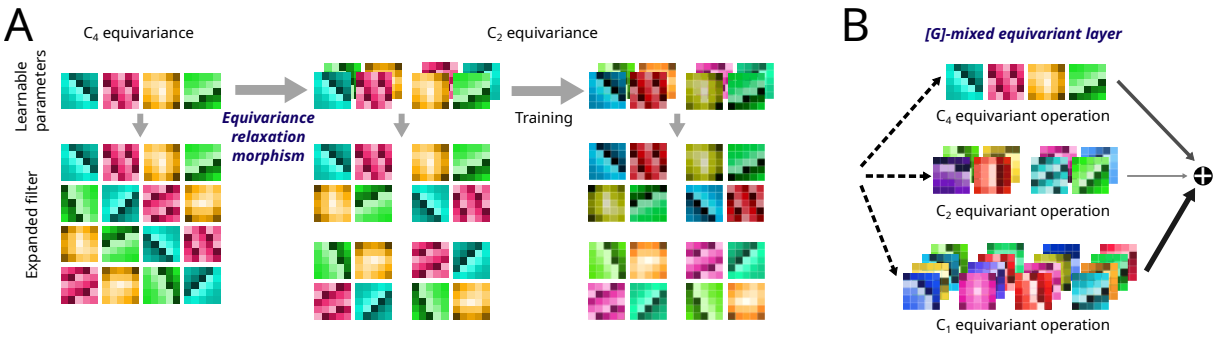


Figure 8: Visualizations of (A) the equivariance relaxation morphism and (B) the $[G]$-mixed equivariant layer, using the C_4 group. In (A), the learnable parameters of a C_4-equivariant convolutional layer are expanded using the group element actions, such that the expanded filter can be used in a standard convolutional layer. Applying the equivariance relaxation morphism reparameterizes the layer to only be architecturally constrained to C_2 equivariance, initialized to be functionally C_4 equivariant. In (B), convolutional operations equivariant to subgroups of C_4 are summed with learnable architectural weighting parameters.

4.2.2 Group Convolutional Neural Networks

Let G be a discrete group. The lth G-equivariant group convolutional layer [196] of a group convolutional neural network (G-CNN) convolves the feature map $f\colon G \to \mathbb{R}^{C_{l-1}}$ output from the previous layer with a filter with kernel size k represented as learnable parameters $\psi\colon G \to \mathbb{R}^{C_l \times C_{l-1}}$. For each output channel $d \in [C_l]$, where $[C] := \{1, \dots, C\}$, and group element $g \in G$, the layer's output is defined via the convolution operator[1]:

$$[f \star_G \psi]_d(g) = \sum_{h \in G} \sum_{c=1}^{C_{l-1}} f_c(h) \psi_{d,c}(g^{-1}h). \tag{2}$$

The first layer is a special case: the input to the network needs to be lifted via this operation such that the output feature map of this layer has a domain of G. In the case of image data, an image x with C channels may be interpreted as a function $x\colon \mathbb{Z}^2 \to \mathbb{R}^C$ mapping each pixel in coordinate space to a real number for each channel, where the cth channel of x is referred to as x_c. The input is $x\colon \mathbb{Z}^2 \to \mathbb{R}^{C_0}$, so the layer is instead a *lifting* convolution:

$$[x \star_G \psi]_d(g) = \sum_{y \in \mathbb{Z}^2} \sum_{c=1}^{C_0} x_c(y) \psi_{d,c}(g^{-1}y). \tag{3}$$

We present our contributions in the group convolutional layer case, although similar claims apply for the lifting convolutional layer case.

4.3 Towards Architectural Optimization over Subgroups

We propose two mechanisms to enable search over subgroups: the equivariance relaxation morphism and a $[G]$-mixed equivariant layer, shown in Figure 8. The proposed morphism, described in Section 4.3.1, changes the equivariance constraint from one group to another subgroup while preserving the learned weights of the initial group convolutional operator. The $[G]$-mixed equivariant layer, presented in Section 4.3.2, allows for a single layer to represent equivariance to multiple subgroups through a weighted sum.

[1]We identify the correlation and convolution operators as they only differ where the inverse group element is placed and refer to both as "convolution" throughout this work.

4.3.1 Equivariance Relaxation Morphism

The *equivariance relaxation morphism* reparameterizes a G-equivariant group (or lifting) convolutional layer to operate over a subgroup of G, partially removing weight-sharing constraints from the parameter space while maintaining the functionality of the layer.

Let $G' \leq G$ be a subgroup of G. Let R be a system of representatives of the left quotient (including the neutral element), so that $G' \setminus G = \{G'r \mid r \in R\}$, where $G'r := \{g'r \mid g' \in G'\}$. Given a G-equivariant group convolutional layer with feature map f and kernel ψ, we define the relaxed feature map $\tilde{f}\colon G' \to \mathbb{R}^{(C_{l-1}\times|R|)}$ and relaxed kernel $\tilde{\psi}\colon G' \to \mathbb{R}^{(C_l\times|R|)\times(C_{l-1}\times|R|)}$ as follows. For $c \in [C_{l-1}]$, $s,t \in R$, $d \in [C_l]$:

$$\tilde{f}_{(c,s)}(g') := f_c(g's), \tag{4}$$

$$\tilde{\psi}_{(d,t),(c,s)}(g') := \psi_{d,c}(t^{-1}g's). \tag{5}$$

We define the *equivariance relaxation morphism* from G to G' as the reparameterization of ψ as $\tilde{\psi}$ (Eq. 5) and reshaping of f as $\tilde{f}$ (Eq. 4). We will show that the new output layer, $[\tilde{f} \star_{G'} \tilde{\psi}]_{(d,t)}(g')$, is equivalent to $[f \star_G \psi]_d(g't)$ down to reshaping. Since the mapping $G' \times R \to G$, $(g', t) \mapsto g't$, is bijective, every g can uniquely be written as $g = g't$ with $g' \in G'$ and $t \in R$. For $g \in G$, $G'g \in G' \setminus G$ has a unique representative $t \in R$ with $G'g = G't$, and $g' := gt^{-1} \in G'$. By the same argument, $h \in G$ may be written as $h = h's$ with unique $h \in G'$ and $s \in R$. With these preliminaries, we get:

$$[f \star_G \psi]_d(g't) = [f \star_G \psi]_d(g) \tag{6}$$

$$= \sum_{h\in G} \sum_{c=1}^{C_{l-1}} f_c(h)\psi_{d,c}(g^{-1}h), \tag{7}$$

$$= \sum_{h'\in G'} \sum_{s\in R} \sum_{c=1}^{C_{l-1}} f_c(h's)\psi_{d,c}(t^{-1}g'^{-1}h's), \tag{8}$$

$$= \sum_{h'\in G'} \sum_{c=1}^{C_{l-1}} \sum_{s\in R} \tilde{f}_{(c,s)}(h')\tilde{\psi}_{(d,t),(c,s)}(g'^{-1}h'), \tag{9}$$

$$= \left[\tilde{f} \star_{G'} \tilde{\psi}\right]_{(d,t)}(g'), \tag{10}$$

which shows the claim. Thus, the convolution of $\tilde{f}$ with $\tilde{\psi}$ is equivariant to G but parametrized as a G'-equivariant group convolutional layer, where the representatives are expanded into independent channels. This morphism can be viewed as initializing a G'-equivariant layer with a pre-trained prior of equivariance to G, maintaining any previous training.

Standard convolutional layers are a special case of group-equivariant layers, where the group is translational symmetry over pixel space. Regular group convolutions are often implemented by relaxation to the translational symmetry group by expanding the kernel via the appropriate group actions, allowing a standard convolution implementation from a deep learning library to be used. The equivariance relaxation morphism generalizes this concept to any subgroup. With the given preliminaries and the case of $G' = T(2)$, $\tilde{f}$ and $\tilde{\psi}$ are computed such that $\tilde{f}_{(c,s)}(g') := f_c(g's)$ and $\tilde{\psi}_{(d,t),(c,s)}(g') := \psi_{d,c}(t^{-1}g's)$ for each $g' \in T(2)$, $c \in [C_{l-1}]$, $s,t \in R$, and $d \in [C_l]$.

Let $S_G := |R|$. The learnable parameters of the G_l-equivariant lth layer with C_l output channels, corresponding to ψ, are stored as a tensor of size $C_l \times C_{l-1} \times S_{G_l} \times K_l \times K_l$. The kernel transformation expands this kernel tensor by performing the action of each $r \in R$ on another copy of the tensor to expand its shape along a new dimension, resulting in a tensor of size $C_l \times S_{G_l} \times C_{l-1} \times S_{G_l f} \times K_l \times K_l$, which is reshaped to $C_l S_{G_l} \times C_{l-1} S_{G_l} \times K_l \times K_l$. The input tensor to the lth layer, corresponding to f, is in the shape of $B \times C_{l-1} \times S_{G_l} \times H_{l-1} \times W_{l-1}$, which is reshaped to $B \times C_{l-1} S_{G_l} \times H_{l-1} \times W_{l-1}$ and convolved with the expanded kernel. The output of shape $B \times C_l S_{G_l} \times H_l \times W_l$ is reshaped to $B \times C_l \times S_{G_l} \times H_l \times W_l$.

To implement the equivariance relaxation morphism, the new kernel tensor is initialized by applying Equation 5 such that result of applying the preceding kernel transformation is equivalent. Our implementation of group actions relies on group channel indexing to represent the order of group elements:

Algorithm 1 Evolutionary equivariance-aware neural architecture search.

procedure EQUINAS$_E$(Initial symmetry group G)
  Initialize population with a G-equivariant group convolutional network.
  for each generation **do**
    for each network in population **do**
      Add children of network with relaxed equivariance constraints into population.
    for each network in population **do**
      Partially train network on dataset.
    Select Pareto-efficient and high accuracy networks as new population.
  return population

to ensure this is consistent before and after the morphism, the appropriate reordering of the output and input channels of the expanded filter are applied upon expansion. The new kernel tensor has a shape of $C_l|R| \times C_{l-1}|R| \times S_{G_l}/|R| \times K_l \times K_l$.

4.3.2 $[G]$-Mixed Equivariant Layer

Towards learning equivariance, we additionally propose partial equivariance through a mixture of layers, each constrained to equivariance to different groups and applied in parallel to the same input then all combined via a weighted sum. The equivariance relaxation morphism provides a mapping of group elements between pairs of groups where one is a subgroup of the other. For a set of groups $[G]$ where each group is a subgroup or supergroup of all other groups within the set, we define a $[G]$-*mixed equivariant layer* as:

$$\left[f \hat{\star}_{[G]} [\psi]\right]_{(d,t)}(g) = \sum_{G \in [G]} z_G \left[f \star_{G'} \widetilde{\psi^G}\right]_{(d,t)}(g) \tag{11}$$

$$= \left[f \star_{G'} \sum_{G \in [G]} z_G \widetilde{\psi^G}\right]_{(d,t)}(g), \tag{12}$$

where each element z_G of $[z] \coloneqq \{z_G | G \in [G]\}$ is an architectural weighting parameter such that $\sum_{G \in [G]} z_G = 1$, G' is a subgroup of all groups in $[G]$, each element ψ^G of $[\psi]$ is a kernel with a domain of G, and $\widetilde{\psi^G}$ is the transformation of ψ^G from a domain of G to G' as defined in Equation 5. Thus, the layer is parametrized by $[\psi]$ and $[z]$, computing a weighted sum of operations that are equivariant to different groups of $[G]$. The layer may be equivalently computed by convolution of the input with the weighted sum of transformed kernels, shown in Equation 12. The implementation for the $[G]$-mixed equivariant layer can be built on top of that of the equivariance relaxation morphism, also using proper input and output channel reordering between layers to ensure correct mixing of group channels.

4.4 Equivariance-Aware Neural Architecture Algorithms

We present two neural architecture search methods that utilize the presented mechanisms for discovering appropriate equivariance during neural network training: Evolutionary Equivariance-Aware NAS (EquiNAS$_E$) and Differentiable Equivariance-Aware NAS (EquiNAS$_D$). Both methods optimize an architecture while learning network weights, returning a final trained network adapted to the equivariances present in the training data. However, they differ in NAS paradigm and approximate equivariance representation: EqufiNAS$_E$, described in Section 4.4.1, searches for networks composed of layers each fully equivariant to possibly different groups, while EquiNAS$_D$, described in Section 4.4.2, searches for smooth mixtures of equivariant layers.

Algorithm 2 Differentiable equivariance-aware neural architecture search.

procedure EQUINAS$_D$(Set of groups $[G]$)
  Initialize network with $[G]$-mixed equivariant layers, parameterized by Ψ and Z.
  while not converged **do**
    Update Z by $\nabla_Z \mathscr{L}(\Psi, Z)$.
    Update Ψ by $\nabla_\Psi \mathscr{L}(\Psi, Z)$.
  return trained network

4.4.1 Evolutionary Equivariance-aware NAS

Towards finding the optimal full equivariance per layer, the equivariance relaxation morphism presented in Section 4.3.1 is applied as the genetic operator in an evolutionary hill-climbing algorithm. The Evolutionary Equivariance-Aware NAS (EquiNAS$_E$) algorithm, given in Algorithm 1, is similar to other evolutionary NAS methods such as Elsken et al. [71] with pareto selection as in Falanti et al. [212]. A population of networks, which starts with an individual with all layers equivariant to the largest possible group, undergoes mutation via equivariance relaxation and selection based on accuracy and parameter count to optimize neural architecture while learning network parameters.

In each generation, candidate networks are evaluated based on maximizing validation accuracy and minimizing parameter count: the entire Pareto front is kept, then additional high-accuracy individuals are added if necessary until the desired parent population size is reached. Offspring are generated from each parent separately through mutation using the relaxation morphism. This preserves the weights of the parameterized equivariance during mutation, allowing for the continuous training of networks over evolution through inheritance from parent individuals. Specifically, mutation reduces a single layer's parameterized equivariance to a subgroup within the constraint that each layer has parametrized equivariance to a subgroup of all preceding layers. This constraint yields local equivariance properties for the network, as shown in Weiler and Cesa [200] and Elsayed et al. [213] to be empirically favorable in image classification tasks. The resulting individuals are each trained independently for a given training time, and then this process repeats.

The second objective of minimizing parameter count is intended to advance efficient networks, such as those with large symmetry groups. Accuracy-based selection alone would necessarily prefer larger networks as mutation via the equivariance relaxation morphism results in two networks with identical performance but different size, the relaxed network having more parameters, until training; potentially short-term increases in validation accuracy after training would then result in the selection of individuals with more parameters. Thus, the proposed strategy of selecting both pareto-front and high-accuracy individuals is intended to maintain a diverse yet efficient population without succumbing to overly greedy selections too early.

4.4.2 Differentiable Equivariance-aware NAS

In a contrasting paradigm, the $[G]$-mixed equivariant layer presented in Section 4.3.2 allows for smoothly searching across a spectrum of equivariance for each layer via a differentiable NAS algorithm. Our Differentiable Equivariance-Aware NAS (EquiNAS$_D$) algorithm, defined in Algorithm 2, is inspired by DARTS [19] with significant changes detailed in the following paragraphs. EquiNAS$_D$ simplifies the bilevel optimization of the architecture weighting parameters Z and kernel weights Ψ into alternating independent updates, computing the gradient update for Z with the current, rather than optimal, Ψ for the current architecture encoded by Z, to boost search efficiency with minimal performance loss compared to higher order approximations [19].

In most differentiable NAS search spaces, the desired output architecture is discretized to select a subset of architectural options within constraints, then the weights are re-initialized and trained within the static architecture. In our formulation, this is not necessary as any mixed operation can be equivalently expressed as a single layer equivariant to any group G' that is a common subgroup to all groups of

the mixed operation (Eq. 12): in our experimental case, this is a standard translation-equivariant convolutional layer, so the final model can be equivalently expressed as a standard convolutional model with encoded partial equivariance. Thus, the final optimized architecture and trained weights are output from the single search process. We explore the standard NAS paradigm, where only the architecture is output and reused for evaluation such that the weights are retrained in this static architecture, in further experiments.

In order to enforce that the scaling of each kernel does not confound the architecture weighting parameters, we use the weight normalizing reparameterization [214] and do not update the scalar norm parameter of each kernel after initialization.

We do not use disjoint datasets for updating Ψ and Z, but rather draw one batch for Ψ and another for Z independently and randomly from the same training split. This allows for a standard dataset split and to use the validation set for hyperparameter tuning.

These two NAS approaches present adaptations of two standard types of NAS, evolutionary and differentiable, to the search for optimal partial equivariance. The equivariance relaxation morphism and the $[G]$-mixed equivariant layer that enable the evolutionary and differentiable search methods respectively are the main focus; the other characteristics of the NAS methods are adapted from existing methods, but further study could advance specialization of equivariance-aware NAS. We next study empirically the two EquiNAS methods on three datasets, one with known rotational symmetry and two with unknown but visually significant rotational and reflectional symmetry.

4.5 Experiments

We focus on the regular representation of groups and show experiments with reflectional and up to 4-fold rotational symmetry groups applied to image classification tasks. Examples of symmetry groups acting on pixel space, which corresponds to $\mathbb{Z}^2$, include $T(2)$, which consists of discrete translations in both dimensions; the cyclical groups C_n, which consist of n-fold rotations; and the dihedral groups D_n, which consist of reflections with n-fold rotations, where $n \in \{1,2,4\}$ for exact symmetry without interpolation. The $p4$ group consists of discrete translations and multiples of 90° rotations and may be represented as $T(2) \rtimes C_4$. The $p4m$ group consists of discrete translations, reflections, and multiples of 90° rotations and may be represented as $T(2) \rtimes D_4$. As standard convolutional layers are already equivariant to $T(2)$, we refer to layers also equivariant to n-fold rotations with or without reflections as D_n or C_n-equivariant, respectively. So, a C_1 equivariant convolutional layer is a standard translation-equivariant convolutional layer. We use $\{C_1, D_1, C_2, D_2, C_4, D_4\}$ as the set of potential groups for mutation in EquiNAS$_E$ and as $[G]$ in EquiNAS$_D$.

We present experiments on image classification for a variety of datasets. The Rotated MNIST dataset [215, rotMNIST] is a version of the MNIST handwritten digit dataset but with the images rotated by any angle. This task serves as a simple investigational study with known symmetry, while the following two tasks are more realistic and complex. The Galaxy10 DECals dataset [216, Galaxy10] contains galaxy images in 10 broad categories. The ISIC 2019 dataset [217–219, ISIC] contains dermascopic images of 8 types of skin cancer plus a null class. For Galaxy10 and ISIC, we down-sample the images to 64×64 due to computational constraints, which adds notable difficulty to the tasks. These tasks exhibit varying levels of rotational and reflectional symmetry, motivating architectural optimization to determine the most effective application of equivariance constraints.

Across all experiments, the architectures are designed to have consistent channel dimensions once expanded to a standard translation-equivariant convolutional layer for each layer across models. Thus, constrained equivariance to a larger symmetry group results in fewer learnable parameters. A layer constrained to C_4 equivariance has $|C_4 \setminus D_4| = 2$ times as many independent channels and as many

parameters as a layer constrained to D_4 equivariance. This is a notably different paradigm than other works that equate parameter counts across architectures with different equivariance properties.

As baseline comparisons, we train and test G-CNNs with static architectures. In addition to the static baselines, we re-implement the residual pathway priors (RPP) approach by Finzi et al. [202] as a C_1 equivariant layer with regularization in parallel with a D_4 equivariant convolutional layer.

Architecture backbone For both EquiNAS$_E$ and EquiNAS$_D$ experiments, we use the same backbone architecture, such that the static baselines have the same architecture across experiments. The architectures have a lifting layer followed by 7 group convolutional layers, for a total of 8 convolutional layers. After 4 layers, the channel count doubles, from 16 to 32 for a D_4 equivariant layer and scaling up for smaller symmetry group equivariance constraints. An average pooling layer is placed after every other layer for all architectures and additionally after the fifth and seventh convolutional layers for Galaxy10 and ISIC. After the final group convolutional layer is a group-dimension average pooling followed by two linear layers to the output dimension. Every convolutional and linear layer except the output layer is immediately followed by a batchnorm then a ReLU.

Hyperparameters The hyperparameters for each algorithm are selected such that baselines only differ by training time and optimizers. The learning rates were selected by grid search over baselines on rotMNIST. For all experiments in Section 4.5.1, we use a simple SGD optimizer with learning rate 0.1 to avoid confounding effects such as momentum during the morphism. For EquiNAS$_E$, the parent selection size is 5, the training time per generation is 0.5 epochs, and the number of generations is 50 for all tasks. Baselines were trained for the equivalent number of epochs. For all experiments in Section 4.5.2, we use separate Adam optimizers for Ψ and Z, each with a learning rate of 0.01 and otherwise default settings. The total training time is 100 epochs for rotMNIST and 50 epochs for Galaxy10 and ISIC. For RPP, we use a C_1-equivariant layer with an $L2$ regularization parameter of 1×10^{-6} in parallel with a D_4-equivariant layer without regularization.

For rotMNIST, we use the standard training and test split with a batch size of 64, reserving 10% of the training data as the validation set. For Galaxy10, we set aside 10% of the dataset as the test set, reserving 10% of the remaining training data as the validation set. For ISIC, we set aside 10% of the available training dataset as the test set, reserving 10% of the remaining training data as the validation. For the latter two datasets, we resize the images to 64×64 due to computational constraints and use a batchsize of 32. The validation sets were previously used for hyperparameter tuning: for experimental results, they are only used for the experiments in Section 4.5.1 as necessary for the EquiNAS$_E$ algorithm. No data augmentation is performed, although the datasets are normalized.

Ablations and Random Search We implement two kinds of random search for each NAS method. The first ablates smart architecture search: EquiNAS$_E$ Random Select works as described in Algorithm 1 but with random parent selection (instead of pareto-front selection) and EquiNAS$_D$ Random Z works as described in Algorithm 2 but with random Z updates (instead of gradient descent) by shuffling Z gradients. The second is more akin to standard NAS random search: for the evolutionary paradigm, we train 30 randomly selected static architectures in the discrete architecture search space for the same training time and selecting the top 5 by validation accuracy, and for the differentiable paradigm, we train 25 randomly selected static architectures in the continuous architecture search space and selecting the top 5 by validation accuracy. 30 and 25 were respectively calculated to be approximately the same compute cost as the trials of EquiNAS$_E$ and EquiNAS$_D$. These are labeled as "Random Static" for both the evolutionary and differentiable paradigms. Since Random Static trains static architectures while EquiNAS$_E$ and EquiNAS$_D$ dynamically search for both architectures and parameters, we take the best 5 architectures for each and retrain their parameters from scratch as in the standard NAS

paradigm, labeled as EquiNAS$_E$ Retrain and EquiNAS$_D$ Retrain, respectively, for fair comparison to Random Static.

For each paradigm of experiments, we present their results in the following subsections, with general discussion following in Section 4.6.

4.5.1 Evolutionary Equivariance-aware NAS

The classification test errors are listed in Table 3 and visualized in Figure 9. The advantages of equivariance search methods are most apparent in the Galaxy10 benchmark. While EquiNAS$_E$ outperforms most baselines on rotMNIST and all baselines on ISIC, it has similar performance on both tasks to the D_4 baseline, and some of the final architectures are very similar to the D_4 baseline architecture. However, the D_4 baseline fails at the Galaxy10 task, demonstrating that the same equivariant architecture can not be naively applied to different tasks. Both search methods, EquiNAS$_E$ and RPP, outperform all baseline models on Galaxy10, and by a large margin for EquiNAS$_E$.

The evolutionary progress for the trial on each task is shown in Figure 11: the selected population maintains a fully equivariant network in every generation, except for the final generation in both Galaxy10 and ISIC. For RotMNIST, the final selected population originates from two main lineages, one staying fully equivariant until the last generations and the other diverging from the fully equivariant network midway through, showing that training with dynamically constrained parameterizations can produce performant models.

In addition to the normally initialized static baselines, we also train and test baselines that are initialized with priors to larger symmetry groups. These are implemented by initializing all layers to be constrained to the prior symmetry group, then using the equivariance relaxation morphism on each layer. EquiNAS$_E$ searches for relaxation schedules that yield trained priors on equivariance, while these additional baselines yield untrained priors. The results in Table 3 show that the C_1-equivariant networks generally improve with either equivariance prior, while the C_4 equivariant networks perform better with D_4 equivariance initialization only when the D_4 constrained baselines also work well. The untrained prior methods do not perform as well as EquiNAS$_E$ on rotMNIST, showing the benefit of investing some training time to the constrained equivariance. For the other tasks, the baselines with priors have better performances than their constrained baseline counterparts.

Shown in Figure 9, EquiNAS$_E$ outperforms EquiNAS$_E$ Random Select, showing the benefit of using informed selection to guide the relaxation of equivariance constraints over training. Additionally, EquiNAS$_E$ Retrain outperforms the Random Static baseline, showing that using compute in an informed search is more beneficial than just randomly searching the space of static architecture constraints.

4.5.2 Differentiable Equivariance-aware NAS

The classification test errors are listed in Table 4 and visualized in Figure 13. EquiNAS$_D$ achieves better test accuracy than the other comparable methods on rotMNIST and Galaxy10. Due to differences in training protocol, only comparisons of relative rankings with Table 3 are possible: baseline methods accuracies followed similar patterns to ranking between experimental paradigms, suggesting the benefit of general C_4 equivariance for rotMNIST and Galaxy10 and general D_4 equivariance, including RPP, for ISIC. In this training protocol notably with adaptive optimizers, the results are more consistent across methods and trials.

The architecture weighting parameter dynamics for each trial are shown in Figure 15 for RotMNIST, Figure 16 for Galaxy10, and Figure 17 for ISIC. The general trend of less constrained layers toward the end of the network supports the conjecture of local equivariance being beneficial. However, this effect is less consistent for ISIC with possibly less inherent symmetry: trials on ISIC, the only task where EquiNAS$_D$ did not exceed baselines, had the most varied architectures. The final mixing of

Method	rotMNIST	Galaxy10	ISIC
EquiNAS$_E$	**1.78 ± 0.04**	**20.3 ± 0.9**	**31.0 ± 0.4**
RPP [202]	2.18 ± 0.04	**24.3 ± 2.8**	32.2 ± 1.7
D_4 baseline	**1.78 ± 0.11**	50.8 ± 17.0	32.1 ± 2.4
C_4 baseline	**1.64 ± 0.22**	29.6 ± 5.5	32.9 ± 1.0
C_1 baseline	5.02 ± 1.15	31.6 ± 4.8	33.2 ± 1.5
C_4 (prior: D_4)	1.93 ± 0.05	27.8 ± 5.3	31.9 ± 1.5
C_1 (prior: D_4)	3.40 ± 0.07	25.9 ± 2.3	**31.4 ± 2.6**
C_1 (prior: C_4)	2.96 ± 0.05	30.7 ± 7.1	32.5 ± 1.1

Table 3: Test error percentages (lower is better) across tasks and approaches. Statistics are aggregated over the final selected population of 5 individuals for EquiNAS$_E$ and across 5 random seeds for all other methods. The **best** and **second best** average errors for each task are highlighted.

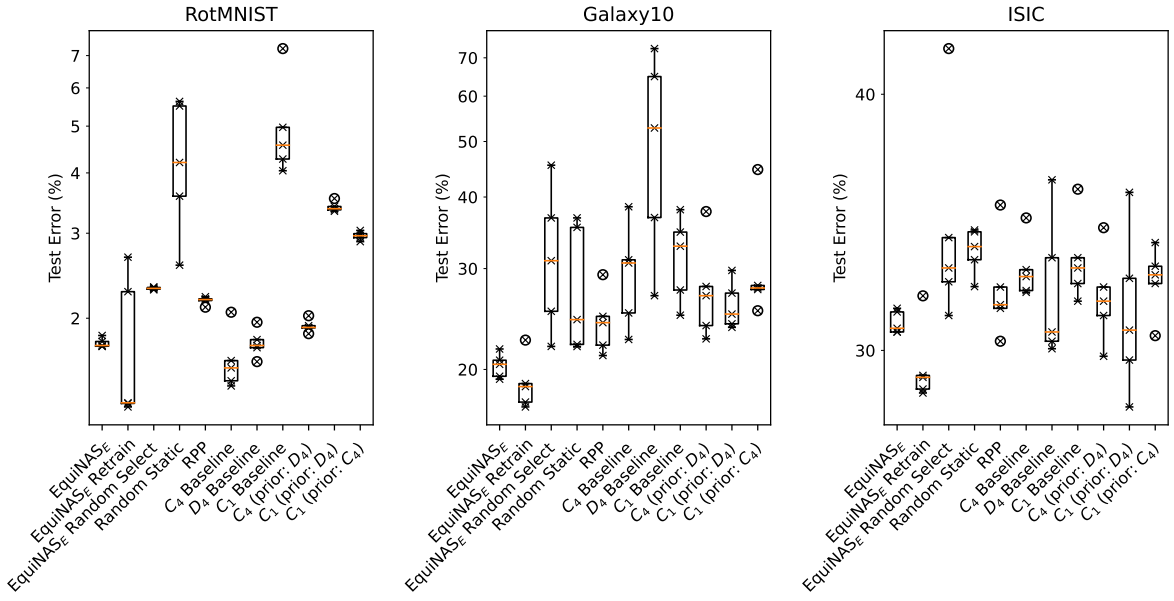


Figure 9: Test errors for experiments of Section 4.5.1, including ablation and random search experiments.

architectures for ISIC included a high level of C_1, indicating that feature analysis outside of these symmetry groups is important for this benchmark.

Previous differentiable NAS works often used regularization of network size or even architecture weighting parameters themselves to encourage efficient architectures with a single highly weighted choice for each layer. However, our algorithm shows strong preference for a single, more equivariant and thus more expressive layer, notably to D_4 or C_4 equivariance, without such regularization. This may be due to the bilevel optimization dynamics: more constrained layers may be able to make more effective updates to more closely approximate the correct gradient computation that assumes optimal weights and thus become favorable compared to the lagging larger layers. This conjecture is shown particularly for trials on Galaxy10: increases in the architectural weighting parameter towards C_1 equivariance often comes after having strong weights for operations equivariant to larger groups.

EquiNAS$_D$ finds competitive architectures on average and can find architectures which outperform baseline choices like architectures fully equivariant to C_1 or D_4. From comparing EquiNAS$_D$ Retrain to EquiNAS$_D$ results in Figure 13, retraining a resulting architecture is not consistently better or worse than using the final weights from EquiNAS$_D$, showing that there isn't a disadvantage to training weights during search and avoiding the additional cost of a post-search training step.

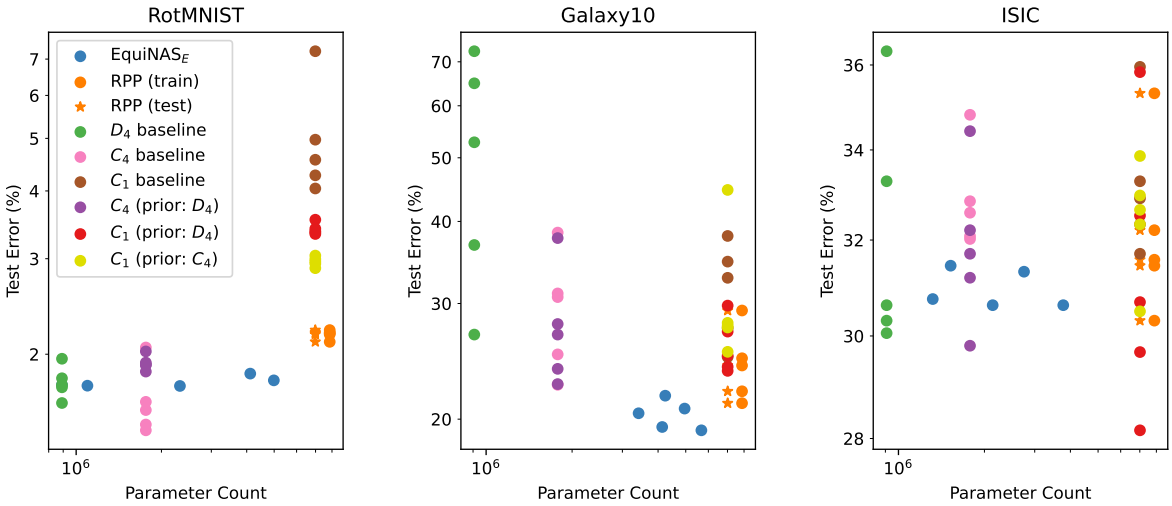


Figure 10: Test errors against stored parameter count for experiments of Section 4.5.1. For EquiNAS$_E$, parameter counts of the final models are shown, although training begins with the same parameter count as the D_4 baselines as shown in Figure 11. Although RPP trains with a higher parameter count, the parameters may be stored in a single filter per layer using Equation 12 for testing.

The use of randomized loss information in Random Static in Figure 13 shows that an informed search for architecture hyperparameters is generally useful. However, experiments on the ISIC benchmark demonstrates that the architecture search can be deceptive and that random loss information can outperform informed loss. This motivates exploration into the use of noise during the search process for architectural parameters.

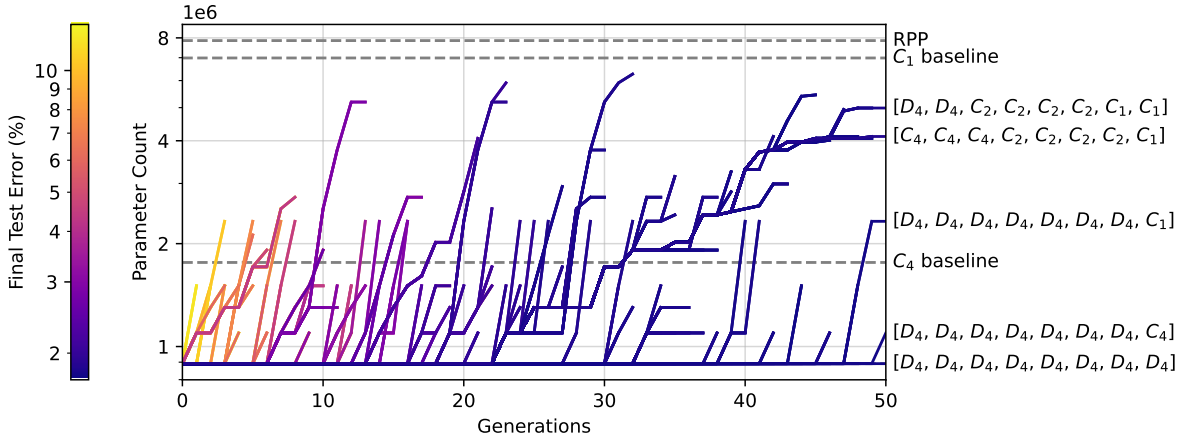
A sampling of random continuous architectures in Random Static in Figure 13 shows that random architectures can perform well on problems where fully equivariant architectures like the D_4 baseline already perform well. However, on the Galaxy10 problem, the D_4 and C_4 baseline have high variance, suggesting that a fully equivariant architecture is sub-optimal. On this baseline, EquiNAS$_D$ greatly outperforms a search of random architectures, demonstrating that EquiNAS$_D$ can discover the appropriate equivariance for a specific dataset over fixed or randomly selected architectures.

The search space for EquiNAS$_D$ is already well-formed for random architectures, compared to the discrete search space of EquiNAS$_E$. This is enabled by the $[G]$-mixed equivariant layer, which is a contribution of this work. Random non-mixed equivariant architectures do worse on all three benchmarks compared to random architectures which use the $[G]$-mixed equivariant layer. This can explain why the EquiNAS$_D$ results are closer to random baselines than the EquiNAS$_E$ results, as the search space permits easily finding the appropriate mix of equivariances compared to a discretized search space.

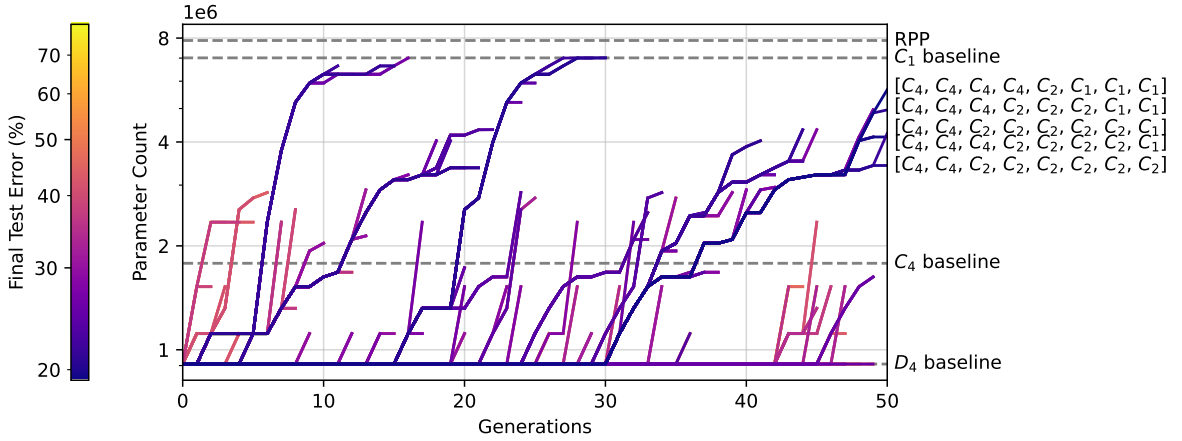
To explore the symmetry discovery of EquiNAS$_D$, we apply it to six augmentations of the MNIST dataset [9], where each augmentation applies the group actions of each group in $\{C_1, C_2, C_4, D_1, D_2, D_4\}$ respectively. The resulting architecture dynamics of this experiment are shown in Figure 12, showing that less augmented versions still have some inherent symmetry, while more augmented versions induce stronger architectural changes towards layers that are equivariant to larger groups. Across all augmentations, earlier layers tended towards more constraints to equivariance.

4.6 Discussion

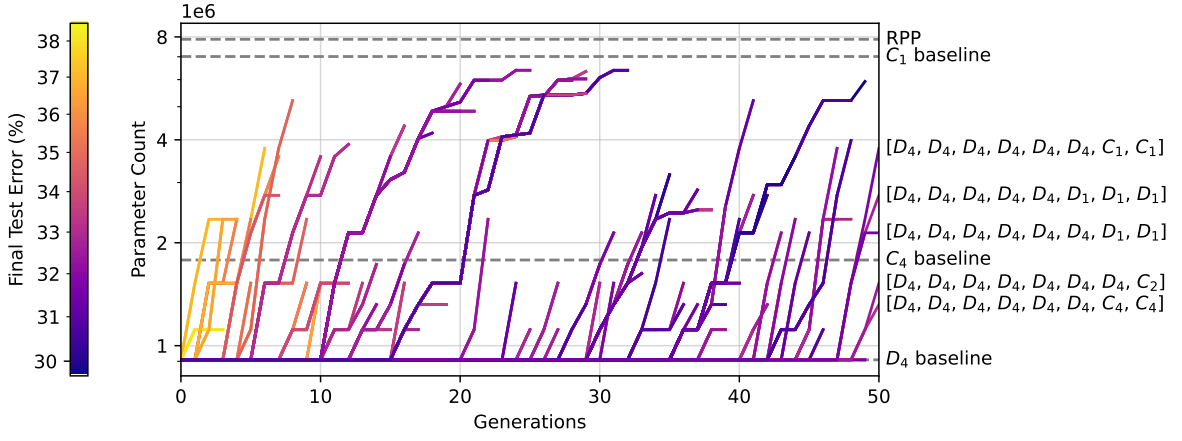
To our knowledge, this is the first work which proposes search methods for networks with dynamically constrained equivariance. Many NAS approaches separately search for an architecture and then reinitialize and retrain the weights, while our two proposed approaches find an optimal architecture with



(a) RotMNIST



(b) Galaxy10



(c) ISIC

Figure 11: Historical parameter counts of each selected individual for EquiNAS$_E$ for each task. The architectures of the final selected population are labeled. Each parameter count history is colored according to the final test accuracy, which is measured of each individual upon removal.

trained weights in a single process, notably with dynamically constrained weights. Gradient-based tuning [220] has shown the benefit not only of optimizing hyperparameters but also of dynamically adjusting them during training [221]. There is an inherent trade-off between accuracy and generalization capabilities with more constrained equivariance: dynamically constrained weights can reap both over the course of training.

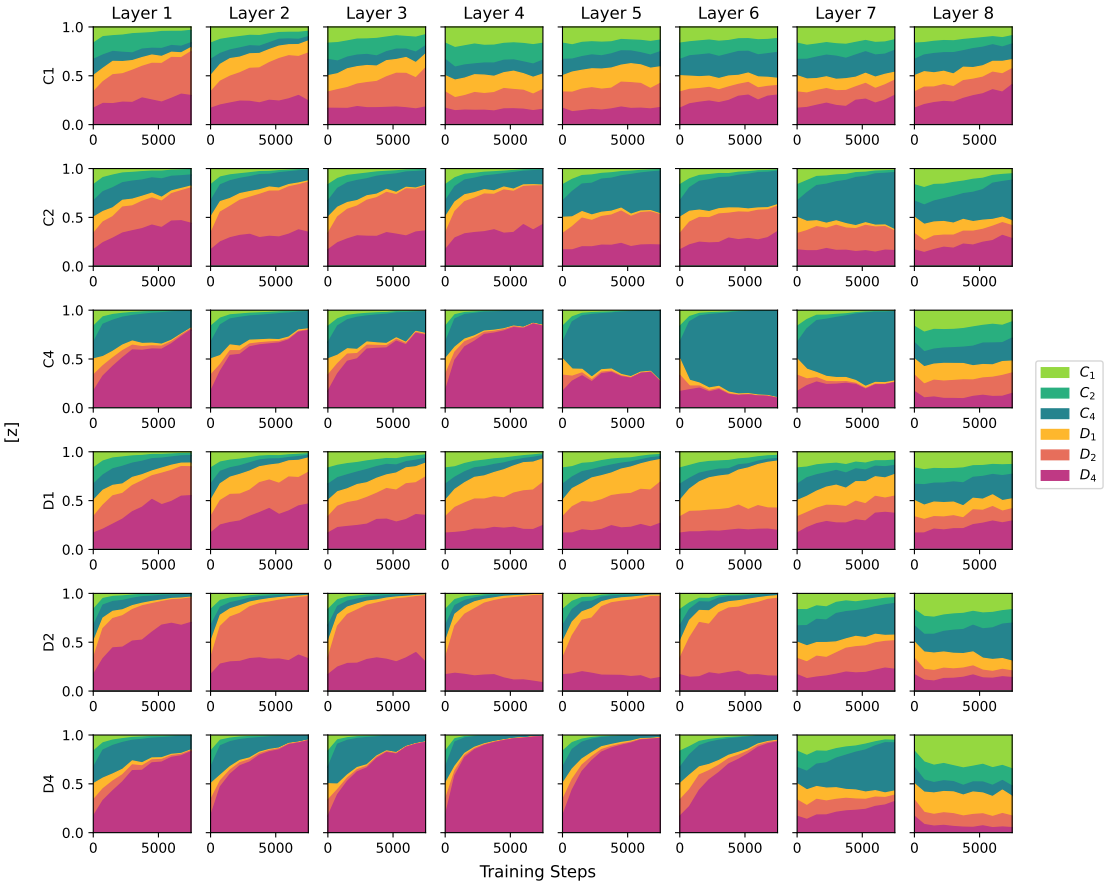


Figure 12: EquiNAS$_D$ on classification of six augmentations of MNIST, where each row is a trial on MNIST augmented to exhibit symmetry to the labeled group.

Method	rotMNIST	Galaxy10	ISIC
EquiNAS$_D$	**2.29 ± 0.27**	**21.8 ± 1.2**	32.8 ± 0.6
RPP [202]	2.89 ± 0.27	**22.0 ± 1.8**	**31.5 ± 0.9**
D_4 Baseline	2.97 ± 1.50	22.5 ± 2.0	**32.0 ± 1.0**
C_4 Baseline	**2.43 ± 0.54**	22.2 ± 2.4	32.8 ± 1.0
C_1 Baseline	3.97 ± 0.75	26.5 ± 1.5	32.9 ± 3.1

Table 4: Test error percentages (lower is better) in percent of incorrect classifications across tasks and approaches. The **best** and **second best** average errors for each task are highlighted.

Our two equivariance-aware NAS approaches have distinct approaches: EquiNAS$_E$ searches for architectures composed of discretely equivariant layers, while EquiNAS$_D$ searches for continuous mixtures of equivariance within each layer. The EquiNAS$_D$ algorithm avoids many known problems in differentiable NAS such as the discretization gap that occurs when searching over a continuous relaxation of a discrete architectural search space [222], such as that of EquiNAS$_E$. Towards searching for discretely equivariant layers using the $[G]$-mixed equivariant layer, proximal NAS algorithms use techniques such as projection [129] and straight-through estimation [223] to avoid the discretization gap and thus may be effective for this application.

The EquiNAS$_E$ algorithm is innately greedy. At each selection step, the population is evaluated on the known current performance rather than the unknown final performance, so this metric is biased to architectures that train quickly. Networks constrained to higher symmetry group equivariance tend to learn faster, but this could be confounded by the equivariance relaxation causing large gradients for

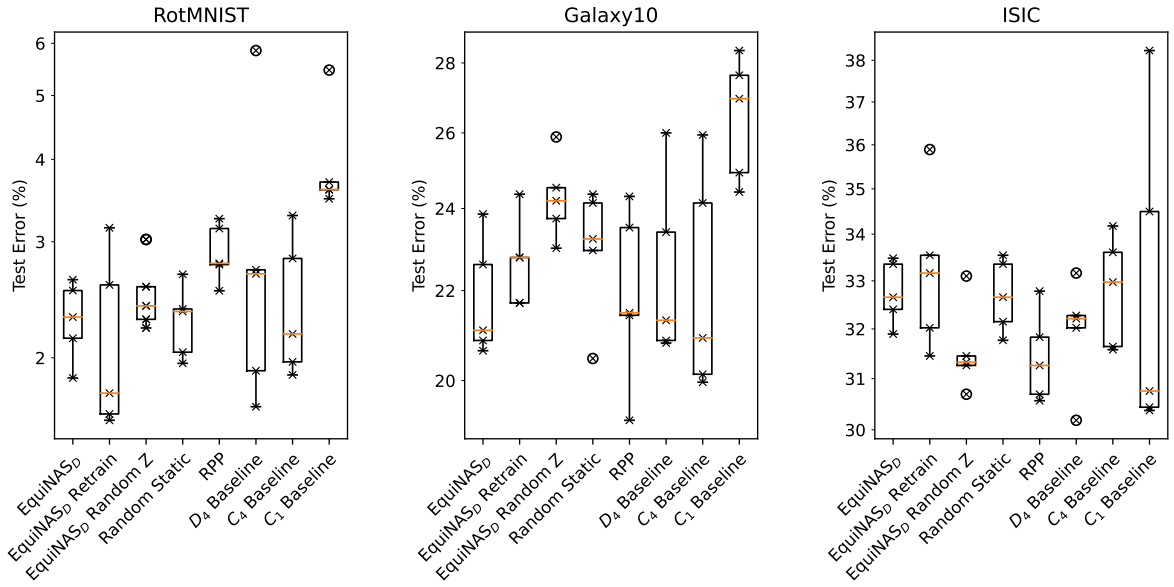


Figure 13: Test errors for experiments of Section 4.5.2, including ablation and random search experiments.

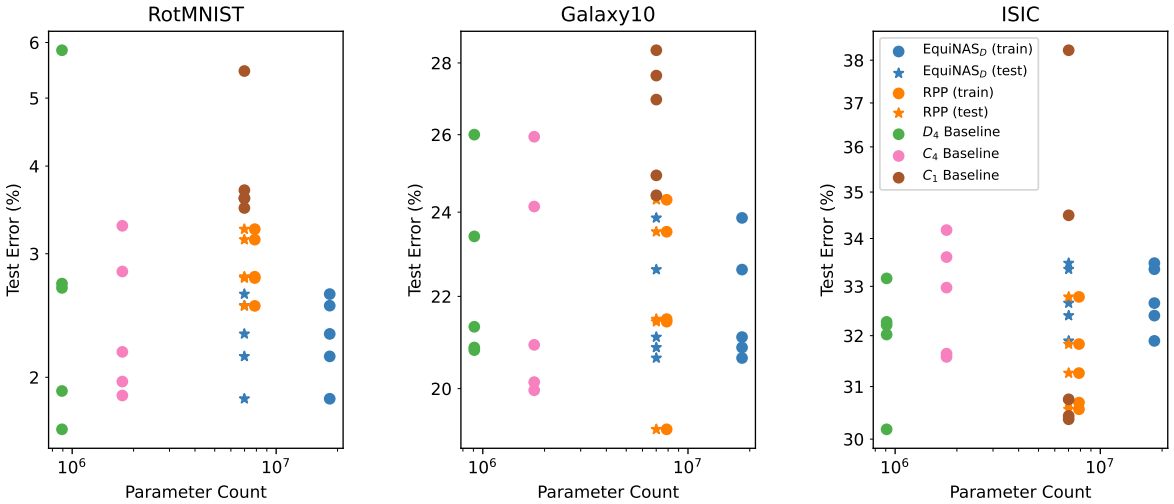


Figure 14: Test errors against stored parameter count for experiments of Section 4.5.2. Although EquiNAS$_D$ and RPP train with a higher parameter count, the parameters may be stored in a single filter per layer using Equation 12 for testing.

the newly unconstrained parameters and thus potentially fast increases in performance. Further work could utilize other metrics for final performance, such as proxies [224].

All of the theoretical and algorithmic contributions of this work are applicable beyond the image classification experiments presented to architectures with parametrized equivariance to any discrete group. We leave the extension to other group representations and domains as future work, such as the continuous case via careful analysis of the regular representation, still given $G' \setminus G$ is finite.

Our proposed equivariance-aware NAS problems can be practically applied to find effective networks for datasets with hypothesizable symmetry. EquiNAS$_E$ may particularly work well on tasks that benefit from local equivariance, which can be determined by analyzing the architecture weighting parameters from first applying the more efficient EquiNAS$_D$, as well as for finding good discrete architectures within which to retrain weights, based on the ablation and random comparisons. For tasks where EquiNAS$_D$ models have less consistent equivariance patterns, EquiNAS$_E$ could be adapted to propose

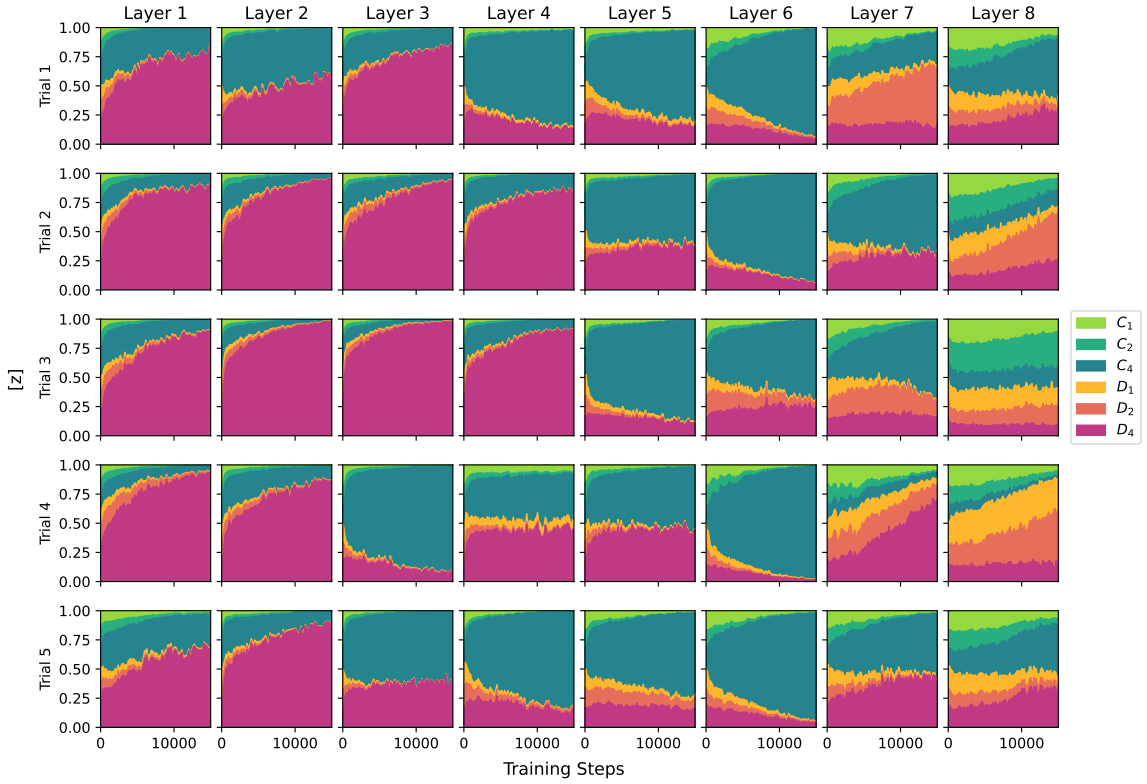


Figure 15: Architecture weighting parameters by layer for all trials on rotMNIST.

candidates that relax any layer in the network, removing our constraint of non-increasing equivariance. We thus recommend EquiNAS$_D$ for practical applications if the final model is not restricted to discrete equivariance, in which case it can be used to inform design decisions for applying EquiNAS$_E$.

Beyond NAS, the equivariance relaxation morphism could be used in other applications such as fine-tuning and distillation. Layers of a pre-trained equivariant network could be expanded via equivariance relaxation before fine-tuning on the same or a new task. Similarly, a network could be distilled to a wider architecture for additional performance benefits.

The proposed equivariance-aware NAS algorithms are intentionally simple to focus on studying the architectural mechanisms presented in Section 4.3, although we have already discussed potential improvements to these algorithms. We include all valid results, even those that do not favor our own techniques, for transparency. This work aims to build a foundation for the intersection of equivariant architectures and NAS, rather than over-engineer the algorithms and experiments for incremental performance gains on selected benchmarks.

Conclusion We present two mechanisms towards equivariance-aware architectural optimization, the equivariance relaxation morphism and the $[G]$-mixed equivariant layer, as well as two NAS algorithms that implement these mechanisms respectively in an evolutionary approach, EquiNAS$_E$, and a differentiable approach, EquiNAS$_D$. We investigate how the dynamic equivariance achieved by these algorithms affects the training and performance of networks across multiple image classification tasks of varying complexity and assumed symmetry, demonstrating that these techniques can search for performant architectures and weights even on noisy tasks. The proposed mechanisms and algorithms are extendable beyond vision tasks to any architecture with parametrized equivariance to any discrete group.

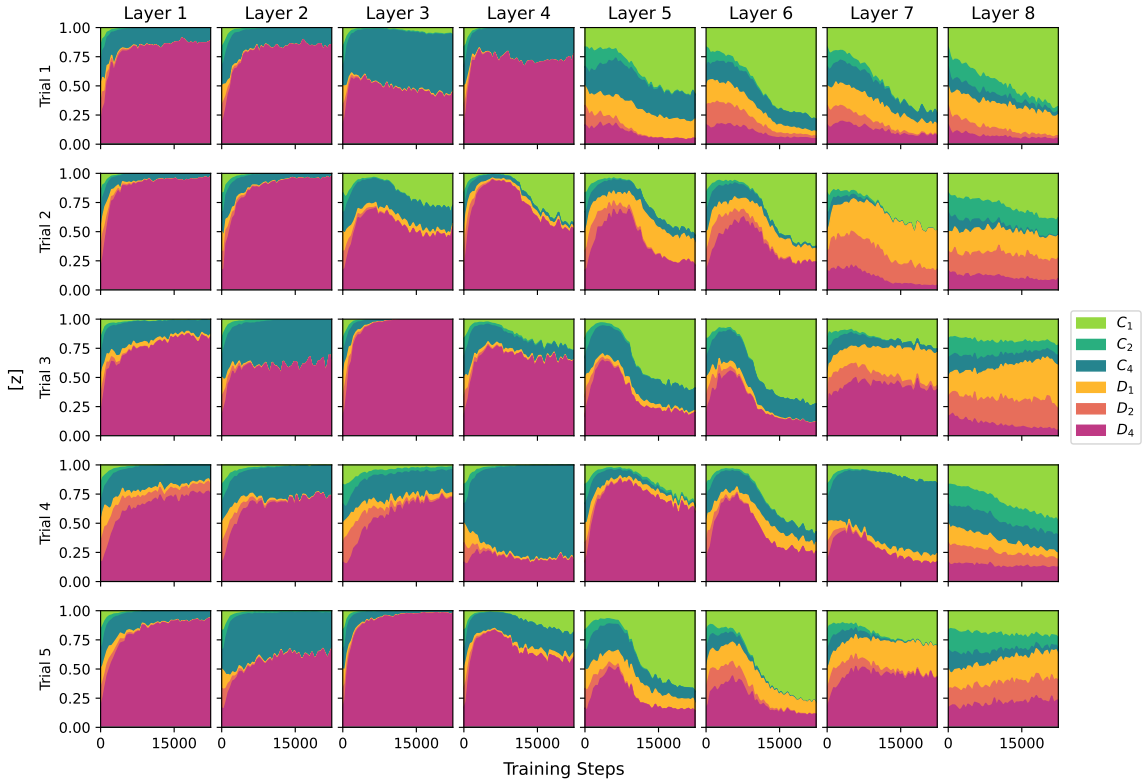


Figure 16: Architecture weighting parameters by layer for all trials on Galaxy10.

The extension of NAS search spaces beyond vanilla convolutions empowers the automation of machine learning, especially for complexly structured data. As convolutions revolutionized computer vision, discovering appropriate architectures that capitalize on both explicit and implicit structure and symmetries in data may potentiate models in much broader domains.

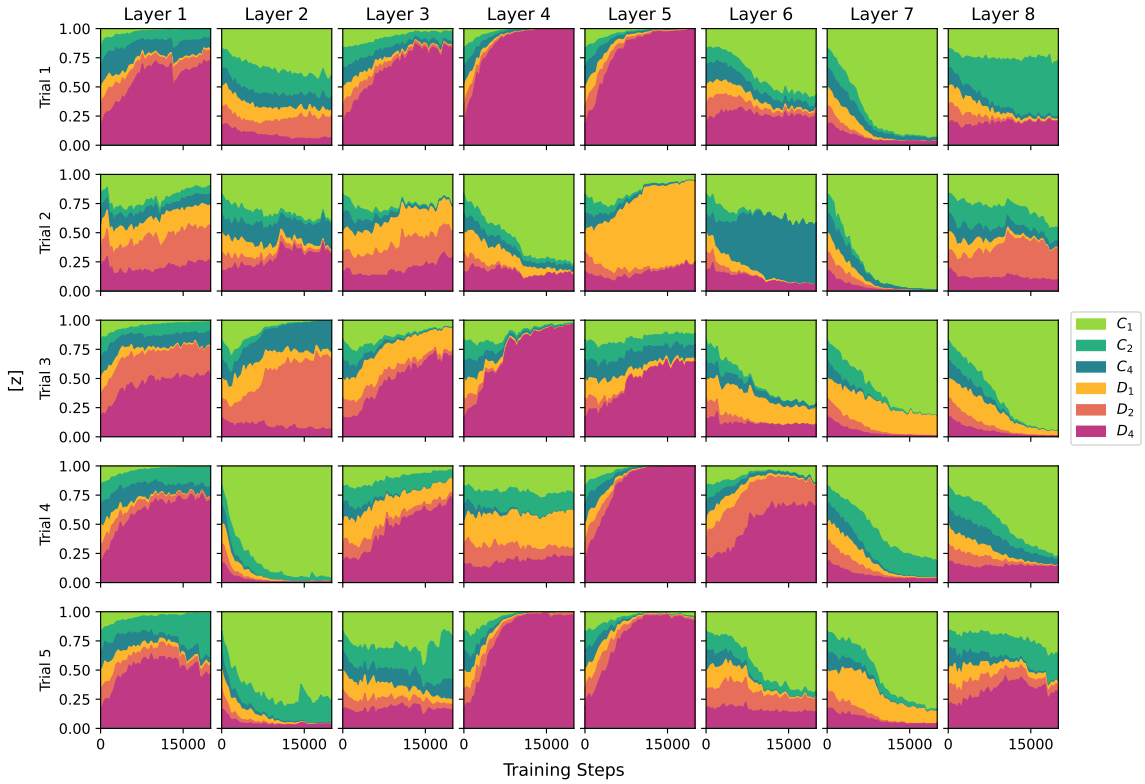


Figure 17: Architecture weighting parameters by layer for all trials on ISIC.

5 Improving Constrained NAS

Neural architecture search (NAS) recently emerged as a path towards automating the structural design of networks. Initial methods used evolutionary algorithms [225] or reinforcement learning [180] to search for networks from building blocks of operators. By relaxing the search space over operators to be continuous, the original Differentiable Architecture Search (DARTS) algorithm was the first to use gradient descent for searching across network architectures [19]. This one-shot search builds a supernetwork with every possible operator at every possible edge connecting activation states within the network, using trainable structural weights across operators within an edge to determine which edges and operators will be used in the final architecture.

While DARTS was able to drastically reduce search time while finding high-performing networks, it suffers from several drawbacks. Among them, contributions pointed out the lack of early search decisions [226–228], failure to approach the final target constraint [161], performance collapse due to skip-connections [229–231], and inefficiency due to unconstrained search [129].

In this work, we posit that difficulties arise in DARTS due to a lack of informed schedule and regularization befitting a constrained bilevel optimization problem. We formulate our contributions in a new augmentation called **DARTS-PRIME** (**D**ifferentiable **Ar**chi**t**ecture **S**earch with **P**roximity **R**egularization and Fisher **I**nfor**me**d Schedule). Specifically, DARTS-PRIME is a combination of two additions to DARTS: a dynamic update schedule, yielding less frequent architecture updates with respect to weight updates, using gradient information to guide the update schedule, and a novel proximity regularization towards the desired discretized architecture, which promotes clearer distinctions between operator and edge choices at the time of discretization.

We analyze these two additions together as DARTS-PRIME, as well as each independently in ablation studies, over standard benchmarks and search spaces with CIFAR-10, CIFAR-100, and Penn TreeBank. We find that these additions independently improve DARTS and combine as DARTS-PRIME to make significant accuracy improvements over the original DARTS method and more recently published variants across all three datasets.

5.1 Background and Related Work

In this section we offer an explanation of the DARTS algorithm formulated as a constrained bilevel optimization problem, focusing on the issues which arise due to this complex search. We also cover related variants of DARTS which propose various solutions to the difficulties arising from this constrained bilevel problem.

DARTS sets up a supergraph neural network that contains all possible options in all possible connections within a framework architecture derived from that of other NAS works [180, 225]. For image classification tasks, this supernetwork consists of repeated cells of two different types, normal and reduce. Each cell of the same type shares the same architecture. This formulation allows for a shallower network with fewer normal cells to be searched over and a deeper one to be used for evaluation, such that both networks fit maximally on a standard GPU. The structure within each cell is a fully-connected directed acyclic graph, receiving input from the two previous cells and using a fixed number of intermediate activation states that are concatenated to produce that cell's output. Each of these edges contains all operators, each of which is a small stack of layers such as a skip-connection or a nonlinearity followed by convolution followed by batchnorm. In order to match the search space to that of the preceding NAS works, the search objective is to find exactly two operators from unique sources, either one of the two previous cells or one of the previous activation states within the cell, to go to each activation state. DARTS accomplishes this by keeping the two strongest operators, as

measured by trained continuous architecture weights, from unique sources for each state within the cell.

The DARTS search can be formalized as follows. Let α_{ijp} be the architecture weight for operator p from node j to node i, so α_{ij} represents the set of architecture weights for all operators from node j to node i and α_i represents the set of architecture weights for all operators from all nodes to node i. The discretization for the DARTS search space may be stated as the intersection of the following sets:

$$S = \bigcap_i S_i^1 \cap S_i^2 \cap S_i^3 \tag{13}$$

$$S_i^1 = \{\alpha_i | \forall\, j \ \text{card}(\alpha_{ij}) \leq 1\} \tag{14}$$

$$S_i^2 = \{\alpha_i | \text{card}(\alpha_i) = 2\} \tag{15}$$

$$S_i^3 = \{\alpha_i | \alpha_{ijp} \in \{0, 1\}\}, \tag{16}$$

where S_i^1 states that there is at most one active operator coming in to each intermediate state from any given source node, S_i^2 states that there are exactly two total active operators coming in to each intermediate state, and S_i^3 states that all α_{ijp} values are binary, so all operators are either active or inactive. Thus, S contains the encoding for all possible discretized architectures α in the search space of DARTS.

During the DARTS search, S is relaxed so that α is not constrained. A softmax operation is applied over continuous α across operators in an edge as an activation. During a forward pass, the output of each edge is computed as the weighted sum of each operator on that edge multiplied by its activated architecture weight; then, the output of each state is computed as the sum of all incoming edges; finally, the output of the entire cell is computed as the channel-wise concatenation of each intermediate state. After searching for a set number of epochs, the softmaxed α is projected onto S to derive the final architecture, as shown in Figure 18.

In order to train the search supernetwork with architecture weights α and network parameters w, DARTS optimizes the following objective:

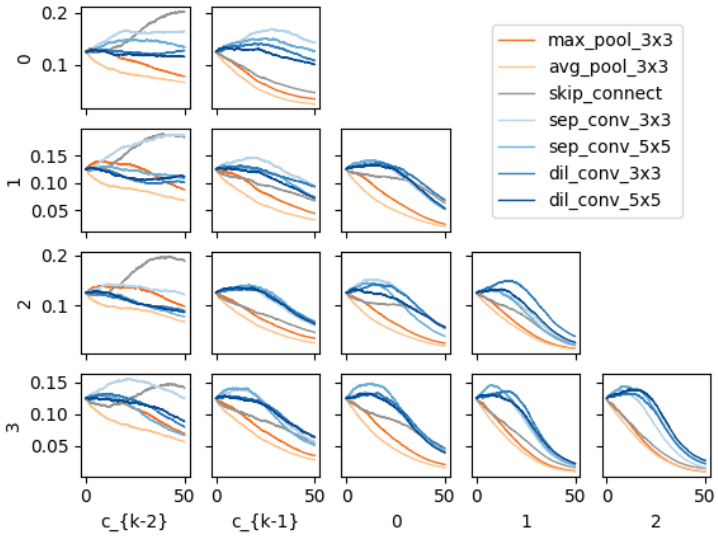
$$\min_{\alpha} \ L_{val}(w^*(\alpha), \alpha) \tag{17a}$$

$$\textbf{s.t.} \ w^*(\alpha) = \arg\min_{w} L_{train}(w, \alpha), \ \alpha \in S. \tag{17b}$$

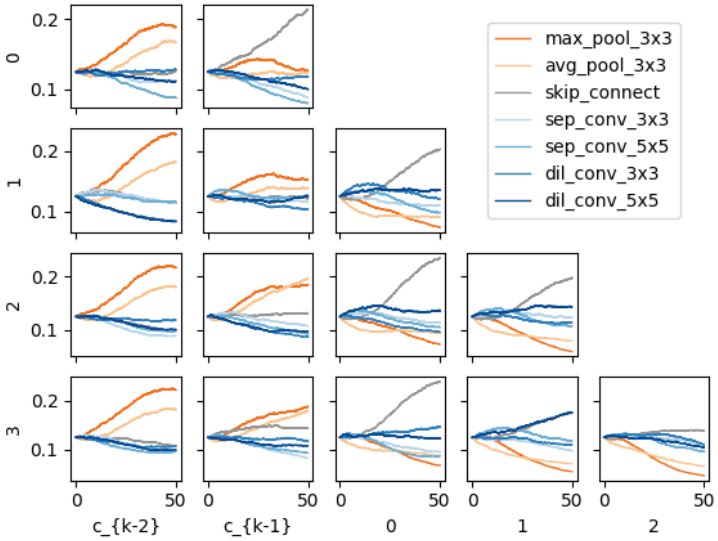
Even with the relaxed space of α, the remaining bilevel optimization problem cannot be directly optimized with standard techniques due to the gradient of the outer optimization (17a) depending on the gradient of the inner optimization (17b), so it is not computationally feasible. Instead, DARTS alternates a single step of gradient descent for α, the outer variable, using either first-order or second-order approximation of the gradient with a single step of gradient descent for w, the inner variable. This naive schedule of alternating steps may not be optimal, even in practice: if the parameterized operators are not trained enough at the current architecture encoding for their parameters to be a reasonable approximation of an optima, their architecture weights will be discounted unfairly. Architectures with more skip-connections converge faster even though they tend to have a worse performance at convergence [230], so this naive schedule may unfairly and undesirably favor skip-connections.

Since the publication of DARTS [19], many variations have been proposed to improve aspects of the original algorithm. We focus on those that are more relevant to the issues we tackle with DARTS-PRIME and study the same DARTS search space in the following.

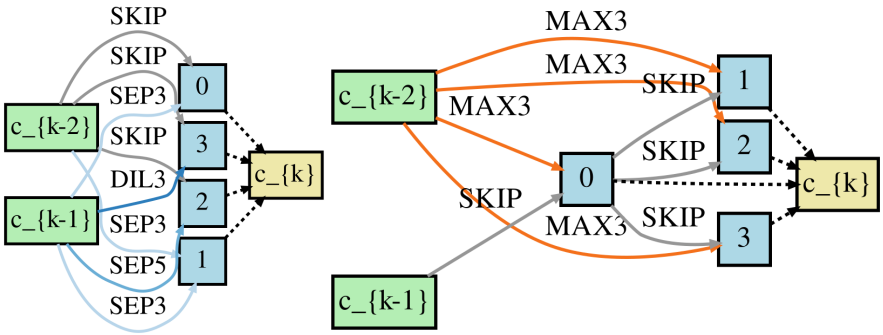
RobustDARTS [231] investigates failure modes of DARTS, including avoiding the collapse towards excessive skip-connections. Skip-connection mode collapse is a known issue with DARTS, and is demonstrated in Figure 19. The authors propose several regularizations of the inner optimization (17b): in particular, their proposed DARTS-ADA utilizes Hessian information to impose an adaptively



(a) Normal cell α over search training



(b) Reduce cell α over search training



(c) Final normal cell

(d) Final reduce cell

Figure 18: Training progression and final results of our highest performing trial of DARTS on CIFAR-10. Each subplot in (a) and (b) represents an edge in the supernetwork cell: each column is the source of the edge, and each row is the destination of the edge. The softmaxed α for each operator is plotted over the search training. To derive the architecture shown in (c) and (d) respectively, the two largest activated α values from unique columns are selected in each row at the end of search. The "none" operator is not shown, as it cannot be selected at discretization.

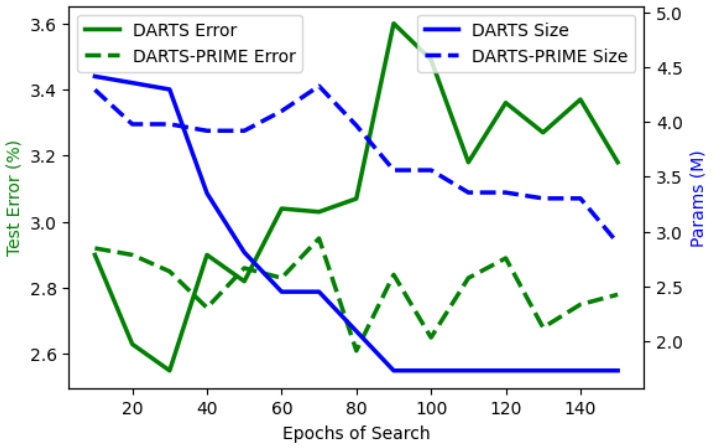


Figure 19: Extended trials of DARTS and DARTS-PRIME on CIFAR-10. A single extended trial for each algorithm was run to 150 epochs, and the architecture checkpointed every 10 generations was evaluated once. In DARTS, the architecture eventually has only skip-connections for the searchable part of the architecture, resulting in a high-error performance. DARTS-PRIME avoids this collapse.

increasing regularization, which bares similarities to both our proposed dynamic informed scheduling and increasing proximity regularization. However, this specific variant is not analyzed in the same standard DARTS search space. DARTS+ [232] uses early stopping to avoid skip-connection mode collapse. DARTS- [229] avoids skip-connection domination by adding an auxiliary skip-connection in parallel to every edge in addition to the skip-connection operation already being included for each edge in the search. The auxiliary skip-connection is linearly decayed over training, similar to our linear schedule of regularization.

NASP [129] tackles the constraint problem in Differentiable NAS with a modified proximal algorithm [233]. A partial discretization is applied before each α step and each w step, so forward and backward passes only use and update the current strongest edges of the supernetwork. While this does improve the efficiency of the search without decreasing benchmark performance, it may lead to unselected operators lagging in training compared to selected ones and thus having an unfair disadvantage during search.

5.2 Towards Constrained Optimization

Neural Architecture Search, particularly when encoded as in DARTS, is a constrained bilevel optimization problem. Both being constrained and being bilevel separately make the optimization less straightforward, such that standard techniques are less applicable. We propose new approaches to the dimensions of bilevel scheduling and regularization towards the constraints to tackle this combined difficult problem in a more dynamic and informed fashion.

5.2.1 Dynamic FIMT Scheduling

Fisher Information of a neural network is closely related to the Hessian matrix of the loss of the network and can be used as a measure of information contained in the weights with respect to the training data as well as a proxy for the steepness of the gradient landscape. While true Fisher Information is expensive to calculate in neural networks, the empirical Fisher Information matrix diagonal can be computed using gradient information already calculated for each batch [234]. This form of the Fisher Information has been used for updating the momentum term in the Adam optimizer [235], overcoming catastrophic forgetting [236], and pruning parameters [237, 238].

We compute the trace of the empirical Fisher Information matrix with each minibatch:

$$\bar{F}(w) = \text{tr}\left(\nabla_w L_{train}(w,\alpha)\nabla_w L_{train}(w,\alpha)^\top\right) \tag{18}$$

Because the network is changing at each weight update and this measure is subject to variability between minibatches, we use an exponentially weighted moving average of the Fisher Information Matrix Trace (FIMT), similar to the decay factor in the Adam optimizer [235]:

$$\bar{F}_n(w) = \lambda\bar{F}(w) + (1-\lambda)\bar{F}_{n-1}(w), \tag{19}$$

for the nth minibatch, where λ is a hyperparameter.

We use the FIMT as a moving estimate of information contained within the weights of the network and the curvature of the gradient landscape. In particular, as noted in DARTS, when the gradient of w is 0, then w is a local optimum and thus $w^* = w$, as desired. As a local optima is approached, the FIMT should decrease and α updates can be triggered more frequently. To achieve this effect, we utilize an exponentially adaptive threshold denoted by h, parametrized by an initial threshold h_0, threshold increasing factor $h_i \geq 1$, expected step ratio r, and moving average weighting factor λ. A threshold decreasing factor $h_i \leq 1$ is computed such that the expected ratio between w and α updates, given constant $\bar{F}_n(w)$, is equal to r. The protocol of this schedule is detailed in 3. The adaptive threshold causes the α update frequency to increase when the FIMT is generally decreasing over batches and decrease when the FIMT is generally increasing over batches. In turn, this adaptive threshold yields a schedule of α and w updates that is more informed about the convergence of the inner optimization given in Equation 17b while maintaining efficiency. This avoids unfair discount of relevant architectures. An example of our dynamic FIMT schedule is shown in Figure 20.

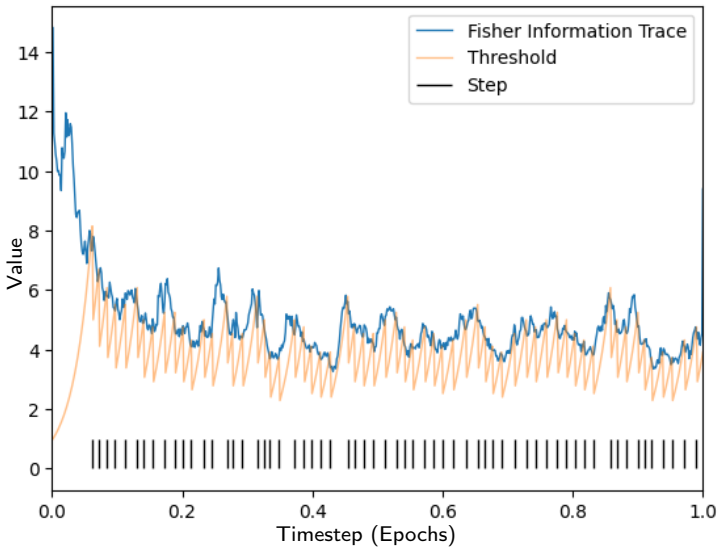


Figure 20: Example of dynamic FIMT schedule, visualized over the first epoch. α updates occur at an interval of roughly every 10 w updates, adjustable by the expected ratio scheduling hyperparameter, but do not occur at the beginning of training when the FIMT is constantly decreasing.

5.2.2 Regularization

In DARTS, the final α values can be far from the resultant discretization, leading to disparity between the search process and its outcome. In order to reward proximity to discretization without losing training signal to inactive portions of the supernetwork, we utilize a proximity regularization that increases in strength over the course of training. The outer objective now becomes

$$\min_\alpha \left(L_{val}(w^*(\alpha),\alpha) + \frac{c\rho_p}{2}\|(\alpha - \Pi_S(\alpha))\|_2\right), \tag{20}$$

Algorithm 3 DARTS-PRIME

Input: Initial threshold h_0, threshold increasing factor h_i, expected step ratio r, moving average weighting factor λ, regularization parameter ρ_p, and learning rates γ_w, γ_α
Output: Discrete architecture $\Pi_S(\alpha)$

Initialize α, w; let $h \leftarrow h_0, h_d \leftarrow \exp(-r \ln h_i), t \leftarrow 1$
while not converged **do**
  $G \leftarrow \nabla_w L_{train}(w, \alpha)$
  $w \leftarrow w - \gamma_w G$
  $\bar{F}_n(w) \leftarrow \lambda \operatorname{tr}\left(GG^{\mathsf{T}}\right) + (1-\lambda)\bar{F}_{n-1}(w)$
  if $\bar{F}_n(w) < h$ **then**
    $\alpha \leftarrow \alpha - \gamma_\alpha \nabla_\alpha \left(L_{val}(w, \alpha) + \frac{c\rho_p}{2}\|(\alpha - \Pi_S(\alpha))\|_2\right)$
    $h \leftarrow h \cdot h_d$
  else
    $h \leftarrow h \cdot h_i$
  $n \leftarrow n+1$
return $\Pi_S(\alpha)$

where c increases linearly from 0 to 1 over the course of search training , ρ_p is a regularization hyperparameter, and Π_S is the projection onto set S, in this case equivalent to deriving a discretized architecture. This formulation allows for more exploration of α and training of w in parameterized operations particularly at the beginning of the search training, progressively increasing the regularization pressure towards discretization as training continues.

5.2.3 DARTS-PRIME

We combine dynamic FIMT scheduling and proximity regularization with DARTS as DARTS-PRIME. The procedure for DARTS-PRIME is given in Algorithm 3.

In addition to these contributions, we also study a regularization based on the Alternating Direction Method of Multipliers (ADMM), an algorithm that uses dual ascent and the method of multipliers to solve constrained optimization problems [239]. We propose as well a non-competitive activation function to naturally permit progressive discretization.

5.2.4 ADMM Regularization

The Alternating Direction Method of Multipliers (ADMM) is an algorithm that uses dual ascent and the method of multipliers to solve constrained optimization problems [239]. ADMM has previously been applied to training sparse networks [240], pruning weights [241, 242], and pruning channels [243], using the version of the algorithm designed for nonconvex constraints such as maximal cardinality of weight matrices. When applied in practice the non-convex objective of neural network objective functions, the optimization variable of the network parameters is usually not evaluated to convergence within every iteration. The implementation of ADMM functions similarly to a dynamic regularization, so we apply it to differentiable NAS in comparison to our proposed proximity regularization.

Directly applying ADMM for nonconvex constraints [239] to the optimization given in Problem 17 using $\alpha \in S$ as the constraint, the iterative solution is

$$\alpha^{k+1} := \arg\min_{\alpha} \left(L_{val}(w^*(\alpha), \alpha) + \frac{\rho_a}{2}\|\alpha - z^k + u^k\|_2^2\right) \tag{21}$$

$$\textbf{s.t. } w^*(\alpha) = \arg\min_{w} L_{train}(w, \alpha) \tag{22}$$

$$z^{k+1} := \Pi_S(\alpha^{k+1} + u^k) \tag{23}$$

$$u^{k+1} := u^k + \alpha^{k+1} - z^{k+1}, \tag{24}$$

where Π_S is the projection onto S, z and u are introduced variables, and ρ_a is a regularization hyperparameter.

Equation 21 is solved by using the DARTS 1st order approximation for n minibatches before updating z and u. The number of w steps (Equation 22) for each α step (Equation 21) is determined by the α schedule.

Because neither level of the bilevel optimization is evaluated to convergence within each iteration, we discount past u values with each update using a discount factor, λ_u. Thus, Equation 24 becomes

$$u^{k+1} := \lambda_u u^k + \alpha^{k+1} - z^{k+1}. \tag{25}$$

5.2.5 CRB Activation

We propose a new activation of α that improves fairness towards discretization and allows for progressive discretization, called Clipped ReLU Batchnorm (CRB) activation. For CRB activation of α, the softmax activation is replaced by unit-clipped ReLU.

This activation function does not enforce all operator weights within an edge to sum to 1. So, the distribution of the output of an edge is not as regulated. To account for this in activation distributions, a non-affine batchnorm layer is placed *after* summing across edges for each state within the cell. This permits the α values across edges to still be comparable, as necessary for discretization. Additionally, the "none" operator can be removed, since the activated α values are no longer dependent on each other.

CRB activation provides bounds within $[0,1]$ and a natural progressive pruning heuristic. Over training, when any architecture weight becomes non-positive, the corresponding operator is pruned from that edge. This progressive pruning allows the network training to focus on the remaining operators and saves computational time.

5.3 Experiments

We test the two presented modifications to DARTS together as DARTS-PRIME as well as each independently as ablation studies. Specifically, we test the dynamic α update schedule using Fisher Information (FIMT), and the proximity regularization (PR), in addition to DARTS-PRIME. We compare these methods to DARTS and its variants on three benchmark datasets: the image classification datasets CIFAR-10 and CIFAR-100, and the text prediction dataset Penn TreeBank.

We also rerun DARTS using the code provided by [19][2]. To better understand the impact of an informed schedule, we study a slight modification of DARTS with a constant schedule of exactly 10 w steps per α step (CS10). The value of 10 was selected for all scheduled experiments as we hypothesize that more w steps should be taken per α step to aid the bilevel optimization approach local minima in w before optimizing α.

5.3.1 Datasets and Tasks

CIFAR-10 and CIFAR-100 [247] are image datasets with 10 and 100 classes, respectively, and each consisting of 50K training images and 10K testing images for image classification. Applying the operator and cell-based search space from [19] to CIFAR-10 has been heavily studied in related NAS works. We additionally perform both search and evaluation on CIFAR-100 using identical protocols and hyperparameters to explore their applicability to similar tasks.

Penn TreeBank [248] (PTB) is a text corpus consisting of over 4.5M American English words for the task of word prediction. We use the standard pre-processed version of the dataset [19, 245]. The DARTS search space for RNNs for text prediction searches over activation functions of dense layers within an RNN cell.

[2] https://github.com/quark0/darts

Task	Method	Test Error (%)	Parameters (M)	Search Time (GPU days)
CIFAR-10	NASNet-A† [180]	2.65	3.3	1800
	AmoebaNet-A*† [225]	2.55 ± 0.05	2.8	3150
	PNAS† [244]	3.41 ± 0.09	3.2	225
	ENAS† [245]	2.89	4.6	0.5
	Random Search [19]	3.29 ± 0.15	3.2	4
	DARTS 1st* [19]	3.00 ± 0.14	3.3	1.5
	DARTS 2nd* [19]	2.76 ± 0.09	3.3	4
	P-DARTS [227]	2.5	-	0.3
	PD-DARTS [226]	2.57 ± 0.12	3.2	0.3
	FairDARTS [161]	2.54 ± 0.05	3.32 ± 0.46	0.4
	R-DARTS (L2) [231]	2.95 ± 0.21	-	1.6
	DARTS- [229]	2.59 ± 0.08	3.5 ± 0.13	0.4
	NASP [129]	2.83 ± 0.09	3.3	0.1
	DARTS reimplementation	2.82 ± 0.19	2.78 ± 0.28	0.4
	+CS10	2.80 ± 0.09	3.41 ± 0.40	0.4
	+FIMT	2.79 ± 0.12	3.49 ± 0.31	0.6
	+PR	2.77 ± 0.03	3.50 ± 0.15	0.5
	DARTS-PRIME	2.62 ± 0.07	3.70 ± 0.27	0.5
	+CRB	3.58 ± 0.79	1.77 ± 0.25	0.4
	+FIMT+CRB	2.91 ± 0.13	2.34 ± 0.35	0.5
	DARTS-PRIME +CRB	2.98 ± 0.22	3.20 ± 0.33	0.5
	+ADMM	2.89 ± 0.05	3.13 ± 0.30	0.4
	+ADMM+FIMT	2.85 ± 0.15	4.43 ± 0.99	0.5
	+ADMM+FIMT+CRB	3.18 ± 0.29	2.06 ± 0.26	0.4
CIFAR-100	P-DARTS [227]	15.92 ± 0.18	3.7	0.4
	DARTS+ [232]	15.42 ± 0.30	3.8	0.2
	R-DARTS (L2) [231]	18.01 ± 0.26	-	-
	DARTS (our trials)	19.49 ± 0.60	2.02 ± 0.15	0.5
	+FIMT	19.44 ± 1.26	2.22 ± 0.22	0.5
	+PR	19.04 ± 0.85	2.24 ± 0.31	0.4
	DARTS-PRIME	17.44 ± 0.57	3.16 ± 0.42	0.5
	DARTS-PRIME +CRB	18.41 ± 0.48	2.28 ± 0.24	0.4

Table 5: Results for CNN architecture search on CIFAR-10 and CIFAR-100. Test errors (lower is better) are listed in percent of incorrect classifications on the test set after retraining each search architecture. Network sizes of the evaluation network are listed in millions of parameters. Search costs are listed in unnormalized GPU days and do not include evaluation costs. Each of our trials were run on a single V100. Mean errors and search costs are listed with standard deviations.
*: Previously reported results where the single best search architecture was evaluated multiple times rather than reporting results across multiple searches. Selecting the best architecture across searches is reflected in the search time.
†: Architectures searched in a slightly different search space in operator types and network meta-structure.

5.3.2 Hyperparameters

We use all protocols and hyperparameters from [19] for each domain of image classification and text prediction and for both search and evaluation unless otherwise stated. We use the first order approximation of the gradient of α. Hyperparameters for CIFAR-100 are maintained directly from CIFAR-10.

When the dynamic FIMT schedule is used, the expected ratio of w steps per α step, r, is set to 10, as in CS10. The initial threshold, h_0, is set to 1.0. The threshold increasing factor, h_i, is set to 1.05. The

Method	Perplexity		Evaluation	Parameters	Search Time
	Valid	Test	Epochs	(M)	(GPU Days)
NAS[†] [180]	-	64	-	25	10000
ENAS[†‡] [245]	60.8	58.6	8000	24	0.5
Random search [19]	61.8	59.4	8000	23	2
DARTS 1st [19]	60.2	57.6	8000	23	0.5
DARTS 2nd [19]	58.1	55.7	8000	23	1
GDAS [246]	59.8	57.5	2000	23	0.4
DARTS (our trials)	68.5 ± 6.2	66.2 ± 6.1	1200	23	0.1
DARTS-PRIME	63.1 ± 2.1	61.1 ± 2.2	1200	23	0.1

Table 6: Results for RNN architecture search on PTB. Performance is measured by perplexity (lower is better). All previous works report the single best result from multiple runs rather than statistics across multiple trials. For our trials, convergence at 1200 epochs was sufficient to differentiate between different methods.
[†]: Architectures searched in a slightly different search space in operator types and network meta-structure.
[‡]: Reevaluated and reported by [19] using the current standard evaluation pipeline.

moving average weighting factor, λ, is set to 0.2. These values are all set to intuitive values without formal tuning.

For proximity regularization (PR), the regularization parameter, ρ_p, is set to 0.1 for image classification and 1.0 for text prediction. This was exponentially searched over 0.001 to 1.0 on CIFAR-10 and PTB, respectively, during early development.

For ADMM regularization, z and u are updated after every 10 α steps. The decay factor, λ_u, is set to 0.8. The regularization parameter, ρ_a, is set to 0.1.

When CRB activation is used, the cosine annealing of the architecture learning rate and the weight decay of α are both turned off.

5.3.3 Data and Training Configuration

After search, we follow the same evaluation procedure as DARTS for image classification and text prediction. After the search training completes and the final discretization selects the cell architecture(s), the evaluation network is reinitialized at the full size, trained on the training set, and tested on the heldout test set.

For trials with the original DARTS schedule of alternating w and α steps, the training and validation split was maintained at 50/50 during search, as done in [19]. For trials with either the constant or dynamic schedule with more w steps per α step, the dataset was split according to the expected schedule of steps. For the ratio of 10 w steps per α step, this results in a dataset split of 90.91% in the training set and 9.09% in the validation set.

A difference from the experiment protocol in DARTS is that we do not select the best architecture from multiple searches based on validation performance. Instead, we evaluate each architecture fully once and report the mean test error across 4 search trials with different random seeds. 4 was chosen as the number of trials due to the prohibitive cost of the evaluation phase which can take up to 3 GPU days per trial on a V100 GPU. While this is a small number of independent trials, we believe that showing the evaluation results from multiple search trials, instead of the best search architecture, is a better measure of search algorithm performance and reliability.

5.4 Results and Discussion

We conduct experiments for each of the contributions independently as well as combined with each other in various configurations in the standard DARTS search space for CIFAR-10. Our experiments also include the original DARTS algorithm. We also apply DARTS, DARTS-PRIME, and a subset of ablation configurations to CIFAR-100 and PTB. The mean and standard deviation across searches of the resulting test error or perplexity, final architecture size, and search time for each are reported in Table 5 for CIFAR-10 and CIFAR-100 and in Table 6 for PTB, compared to reported results from related works. The α progressions over training and final architectures for the best trial on CIFAR-10 are shown in Figure 18 for DARTS and Figure 21 for DARTS-PRIME. The best architectures for DARTS-PRIME are shown in Figure 23.

The DARTS-PRIME trials show significant improvements over DARTS and match state-of-the-art results within the standard CIFAR-10 search space and task while also yielding one of the lowest variances in performance across our ablation studies. The training progress and final cell architectures for the best trial of DARTS-PRIME are shown in Figure 21. This architecture has an evaluation size of 3.76M parameters and a test error of 2.53.

DARTS-PRIME avoids skip-connection mode collapse, as shown in Figure 19. This is important in domains with variance in parameterization across operators. Avoiding skip-connection mode collapse in the CNN search space decreases the need for tuning the length of search, which improves generalizability across tasks.

We see a clear correlation between network size and performance for this search space in image classification tasks, further visualized in Figure 26. This is one of the reasons why the skip-connection mode collapse is catastrophic.

5.4.1 Benefits of Scheduling

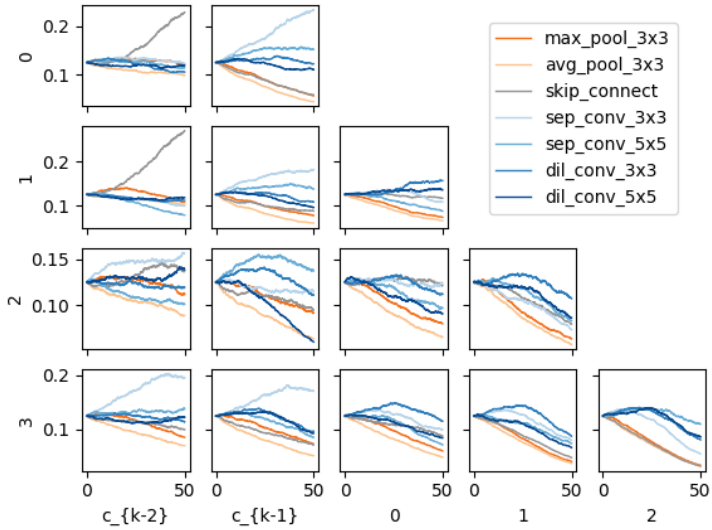
Using either CS10 or dynamic FIMT scheduling increases the network size in all paired experiments and tends to improve performance across datasets. This is tied to avoiding the skip-connection mode collapse. Even with many fewer α updates overall, trials using the less frequent schedules have comparable or larger ranges of α values at discretization. Thus, using more informed architecture steps helps prevent unfair suppression of parameterized operations.

Both the constant and dynamic schedules show similar improvement over the original schedule. One characterization of the dynamic schedule is that in the first epoch, the α updates are delayed significantly beyond the expected schedule, as shown in Figure 20. This appears beneficial, as this gives parameterized operations more time to train before α updates start to occur. Characterizing further benefits of dynamic scheduling over constant scheduling is left as future work. Additionally, other information metrics and dynamic scheduling techniques beyond Fisher Information may be explored.

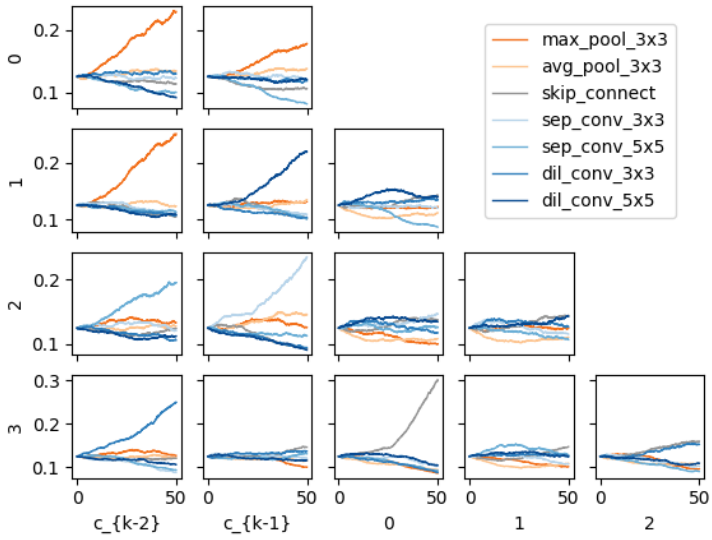
5.4.2 Regularization Improves Reliability

Proximity regularization yields better and more consistent test error and perplexity results, whether applied alone or with FIMT as in our proposed DARTS-PRIME, especially for CIFAR-10. This is very beneficial for one-shot tasks to improve reliability.

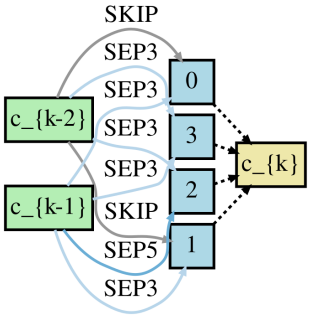
Using ADMM regularization generally improves performance in each paired experiment. However, proximity regularization was more performant in all cases. ADMM is a very powerful algorithm but with many hyperparameters that require careful tuning when applied in practice. This leaves further development as open work.



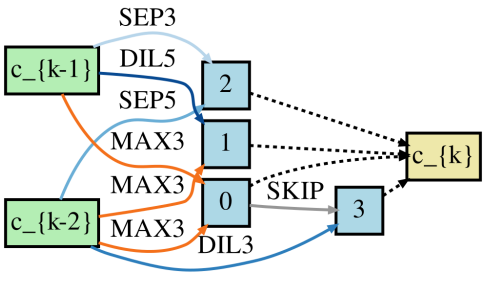
(a) Normal cell α over search training



(b) Reduce cell α over search training

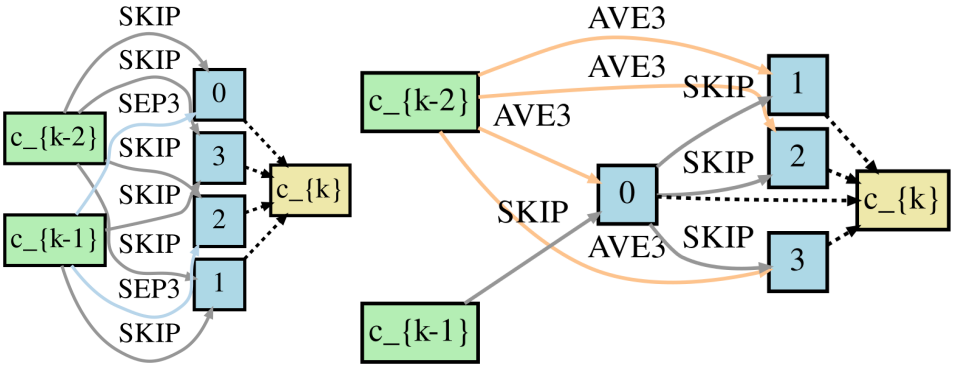


(c) Final normal cell



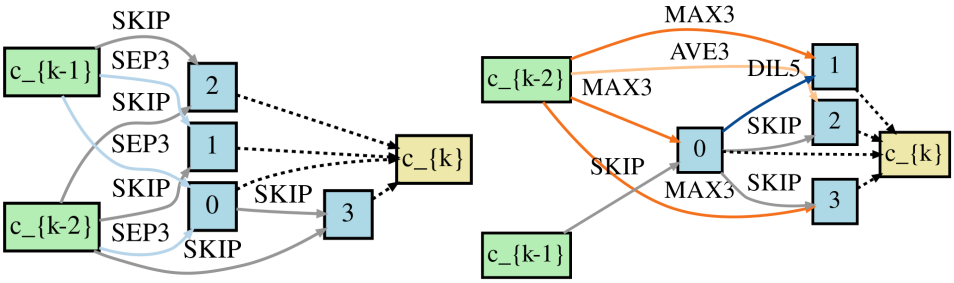
(d) Final reduce cell

Figure 21: Best trial of DARTS-PRIME on CIFAR-10.



(a) Final normal cell **(b)** Final reduce cell

Figure 22: Best final architecture of DARTS on CIFAR-100.



(a) Final normal cell **(b)** Final reduce cell

Figure 23: Best final architecture of DARTS-PRIME on CIFAR-100.

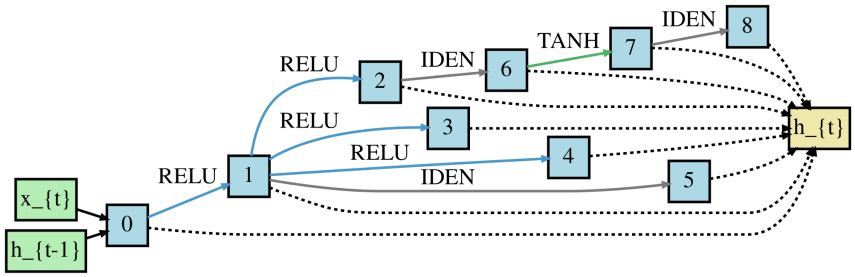


Figure 24: Best final architecture of DARTS on PTB.

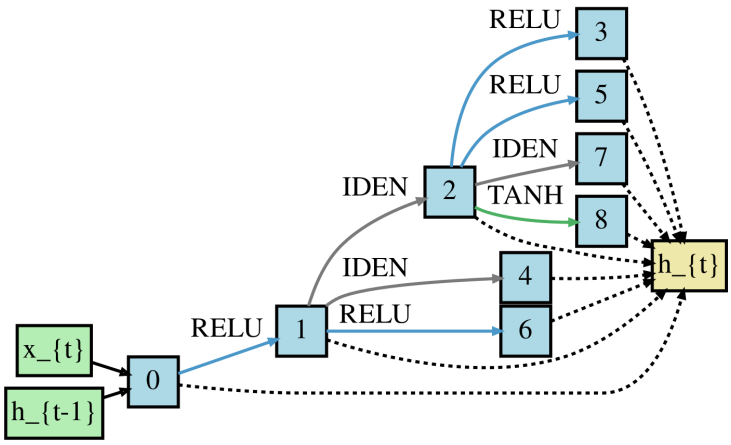
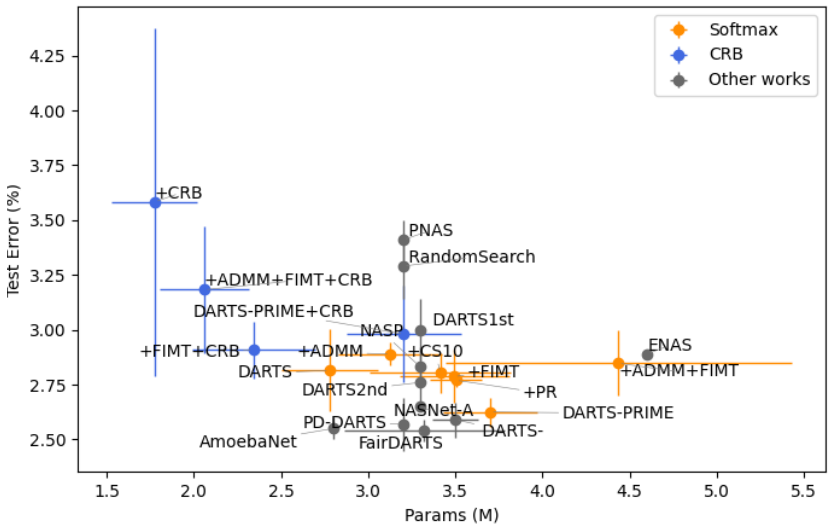
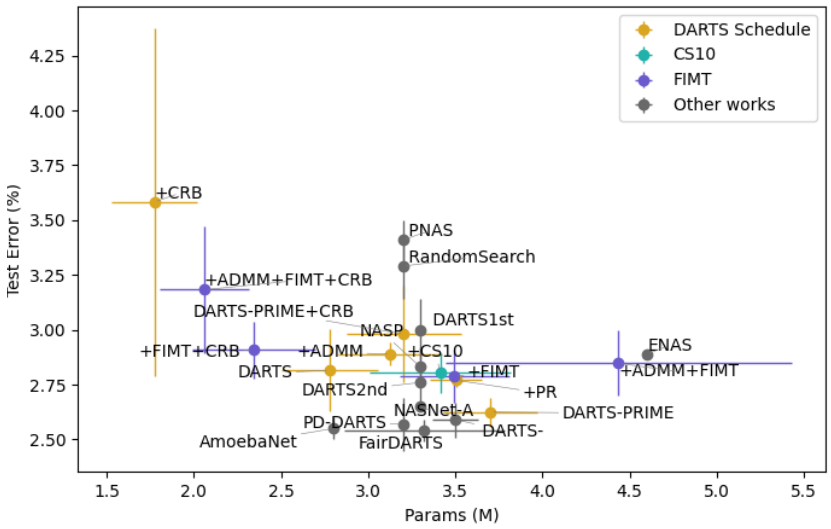


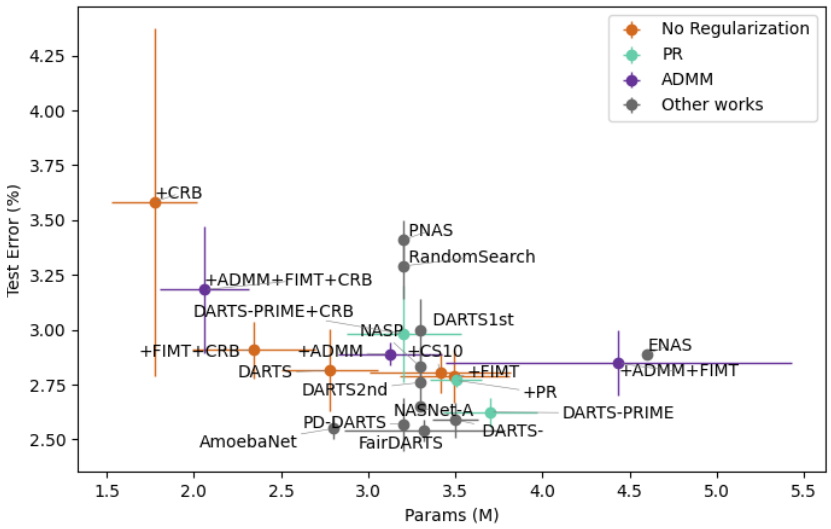
Figure 25: Best final architecture of DARTS-PRIME on PTB.



(a)



(b)



(c)

Figure 26: Comparison across CIFAR-10 trials of test error vs. network size, recolored by (a) activation, (b) schedule, and (c) regularization.

The linear scheduling of the increase of proximity regularization strength is a simple implementation choice that yields sufficient performance improvement. It may conflict with applying early stopping as done in other DARTS variants, but only minorly, in a similar way as standard training protocols such as the cosine annealing of architecture weights used in DARTS. This leaves other informed and dynamic schedules, such as similar to our FIMT schedule, as an open question.

5.4.3 CRB Activation Produces Smaller Networks

For all combinations, the CRB activation finds smaller networks than the corresponding experiment with softmax activation, albeit with a small degradation in performance. This pattern is shown in Figure 26a. We noted that only trials using CRB activation selected any pooling operators in the normal cell architecture, where other methods use convolutional operators. This could cause performance degradation due to the difference in depth between the search and evaluation networks, particularly since only normal cells are added to reach the final depth.

We also note that CRB activation has high variance without regulation or informed scheduling. Particularly in the DARTS+CRB experiment, one trial had a very high error of 4.95, which greatly increased the mean and variance. This trial fell into skip-connection mode collapse. As discussed in the following, scheduling and especially regularization help prevent this collapse from occurring, which is beneficial if the search is applied in a truly one-shot task. On the other hand, if multiple searches can be completed and small network size is beneficial to the application, this protocol may actually be the best approach to apply.

An example of progressive pruning provided by CRB activation is shown in Figure 27(a-b).

5.4.4 DARTS-PRIME Performance Beyond CIFAR-10

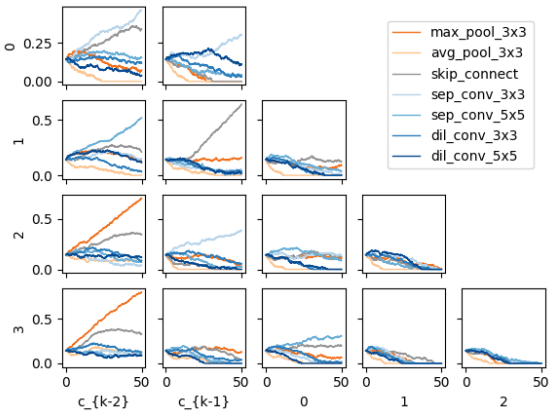
Our findings in CIFAR-10 are also generally supported by the results on CIFAR-100 and PTB: DARTS-PRIME consistently and significantly outperforms DARTS. Specifically on PTB, DARTS-PRIME also has higher reliability in performance. The architecture from our best trial of DARTS-PRIME on CIFAR-100, shown in Figure 23, has an evaluation size of 3.56M parameters and a test error of 16.82. The architecture from our best trial of DARTS-PRIME on PTB, shown in Figure 25, has an validation perplexity of 60.4 and a test perplexity of 58.2 in the shortened evaluation time compared to previous works.

We note that no further hyperparameter tuning was completed for CIFAR-100, but we anticipate that tuning directly on CIFAR-100 may have allowed DARTS-PRIME to get similar low-variance performance results as the other tasks. Only minor tuning was completed for PTB, which differs from CIFAR in application domain and search space, as the operators of the RNN search space are activation functions, which are not parameterized and do not affect the overall size of the network. The performance of DARTS-PRIME on CIFAR-100 and PTB shows the applicability of the methods and hyperparameters within DARTS-PRIME without extensive searching on target datasets and tasks.

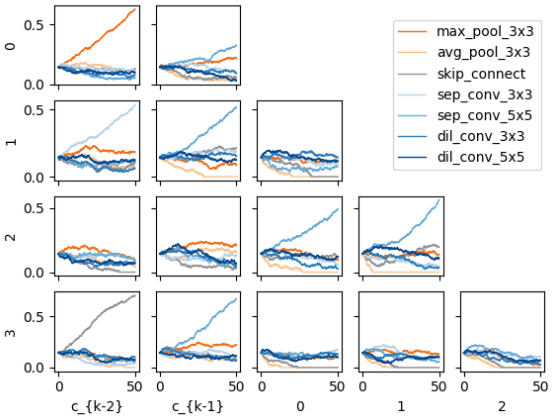
5.5 Conclusion

In this work, we propose DARTS-PRIME with alternative approaches to two dimensions of differentiable NAS algorithms: informed dynamic scheduling of gradient descent steps and regularization towards discretization. Each of these are analyzed with ablation studies to isolate the effects of each approach across domains and tasks. For practitioners, we recommend dynamic FIMT scheduling for improved network performance and proximity regulation for both improved and more reliable network performance.

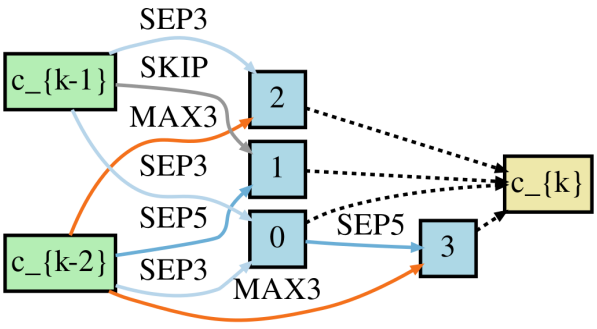
The benefits of these approaches are apparent in results on the CIFAR-10, CIFAR-100, and PTB benchmarks yet also offer insight for different applications of differentiable NAS. The majority of DARTS



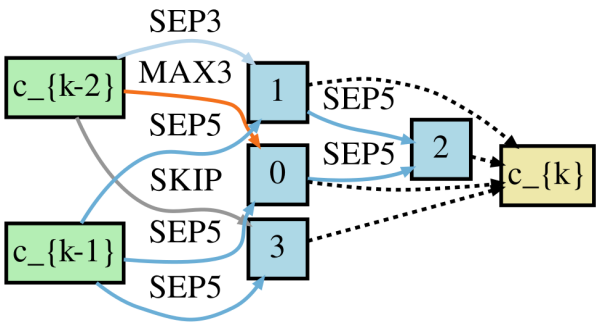
(a) Normal cell α over search training



(b) Reduce cell α over search training



(c) Final normal cell



(d) Final reduce cell

Figure 27: Best trial of DARTS-PRIME +CRB.

variants, including this work, focus on narrow and static benchmark problems, achieving similar values of minimal error or perplexity through extensive weight training and hyperparameter tuning. However, more dynamic tasks, such as shifting data distributions or semi-supervised learning as in reinforcement learning, could highlight the benefits of using dynamic optimization schemes, such as our proposed improvements of scheduling and regularization.

The two improvements of DARTS-PRIME, proximity regularization and a FIMT informed schedule, are modular and could be added to other differentiable NAS algorithms beyond DARTS [19]. In particular, proximity regularization may be applied to any algorithm utilizing a continuous relaxation of the architecture, and dynamic FIMT scheduling may be applied to any NAS method posed as a multilevel optimization. While DARTS-PRIME shows a significant improvement over DARTS, we believe that the contributions of a dynamic FIMT scheduling and proximity regularization can be beneficial for one-shot NAS methods in general.

Dynamic scheduling of structural steps permits more flexible learning algorithms that adapt to the training trajectory and make architectural changes when appropriate. This work contributes towards the challenges of automating machine learning and dynamic adaptation during the learning process, initiating investigation of dynamic scheduling for structural changes. These ideas will be further explored in another structural paradigm, neural creation, in the following section.

6 Neurogenesis

Most existing structural learning approaches such as neural architecture search (NAS) [80] or pruning [78] prune away from a predefined search space, but growing architectures is not yet well understood. By dynamically building ANNs throughout learning, we can aim towards networks which learn not only parameters but also their architectures for specific tasks, potentially improving learning or performance of the final architecture and avoiding expensive hand-tuning of architectures. Furthermore, dynamic ANN architectures can allow continual growth to include more information and tasks as necessary. Dynamic architecture changes may add or remove structural units or connections, operating on the basis of neurons, channels, layers, or other structures. We consider neurogenesis to be the form of structural learning that adds new neurons or channels within existing layers, complementary to structural pruning often used by channel-searching NAS techniques such as [107], [117] and to network deepening such as [146]. Our focus in this chapter is neurogenesis, which is a naturally difficult and less well-studied form of structural learning that invokes multiple questions, specifically when to add neurons, where to add them in the network, and how to initialize the parameters of new neurons; questions which have an undefined search space that must be artificially constrained for optimization.

While considering the trade-off of performance versus efficiency of the network, including training time, inference time, and memory size, we aim to grow ANNs without a preset final size. We propose that an effective neurogenesis algorithm should converge to an efficient number of neurons in each layer during the course of training.

We present the following contributions towards understanding and utilizing neurogenesis.

- We introduce a framework for neurogenesis, decomposing it into triggers and initializations.
- We present novel triggers and initializations based on orthogonality of either post-activations or weights that together form the NORTH* suite of strategies, as well as a gradient-based trigger to complement existing gradient-based initializations. These algorithmically answer when, where, and how to add new neurons to ANNs.
- We compare these contributions with existing initialization methods from the literature and baselines in a variety of tasks and architecture search spaces, showing that NORTH* strategies achieve compact yet performant architectures.

6.1 Background

6.1.1 Problem Statement and Notations

An artificial neural network (ANN) may be optimized through empirical risk minimization:

$$\arg\min_{f} \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim D}\, L(f(\boldsymbol{x}),\boldsymbol{y}) \tag{26}$$

for a neural network f, a loss function L, and a dataset D consisting of inputs x and outputs y. In the simple case of a dense multi-layer perceptron (MLP) with d hidden layers, f may be expressed as $f(\boldsymbol{x}) = \sigma_{d+1}(\boldsymbol{W}_{d+1}\sigma_d(...\boldsymbol{W}_2\sigma_1(\boldsymbol{W}_1\boldsymbol{x})...))$, where σ_l is a nonlinear activation function, adding a row for the bias, and $\boldsymbol{W}_l \in \mathbb{R}^{M_l\times(M_{l-1}+1)}$ is the weight matrix for the l^{th} layer, including the bias parameters. We distinguish between input and output weights of neurons in $\boldsymbol{W}_l$ with M_l rows that each represent the fan-in weights of a neuron in layer l receiving input from each of the M_{l-1} preceding neurons and equivalently $M_{l-1}+1$ columns for which each of M_{l-1} are the fan-out weights of a previous-layer neuron and the last column is the biases.

We define $\boldsymbol{z}_l = \boldsymbol{W}_l \sigma_{l-1}(\boldsymbol{W}_{l-1}\sigma_{l-2}(...\boldsymbol{W}_2\sigma_1(\boldsymbol{W}_1\boldsymbol{x})...))$ as the pre-activations and $\boldsymbol{h}_l = \sigma_l(\boldsymbol{z}_l)$ as the post-activations of layer l. For n samples, i.e. a mini-batch, $\boldsymbol{Z}_l \in \mathbb{R}^{M_l \times n}$ is the pre-activation matrix and $\boldsymbol{H}_l = \sigma(\boldsymbol{Z}_l)$ is the post-activation matrix.

To perform neurogenesis, the addition of k neurons to the l^{th} layer is accomplished by appending k rows of fan-in weights to $\boldsymbol{W}_l$ and k columns of fan-out weights to $\boldsymbol{W}_{l+1}$. Utilizing this operation in the empirical risk minimization of Equation (26) defines the neurogenesis optimization. We restrict the search space of where to add new neurons to within existing layers.

6.1.2 Neurogenesis

We consider neurogenesis through the following dimensions.

- When: Standard learning algorithms naturally discretize the learning process into steps, so neurogenesis may occur at any step of training.
- How many: When neurogenesis is triggered at a step, multiple neurons can be added at once.
- Where: In standard ANN architectures, this amounts to which layer neurogenesis occurs in.
- How: The fan-in and fan-out weights and bias of each new neuron must be initialized.

Most existing neurogenesis works focus on "How", or the initialization of neurons, using a fixed schedule of additions. Some works have approached global scheduling, such as [147, 249, 250] using convergence as a global trigger for neurogenesis. However, this leads to neurogenesis happening later in training and does not explore the questions of "Where" or "How many". We posit that, in order to be competitive with static networks in terms of training cost, neurogenesis should happen throughout training, including before convergence.

To answer "How", most existing non-random neurogenesis initializations in recent literature are gradient-based. Firefly from [110] generates candidate neurons that either split existing neurons with noise or are completely new and selects those with the highest gradient norm. [101] also create noisy copies, selecting from high-saliency neurons. Both GradMax from [142] and NeST from [68] compute an auxiliary gradient of zero-weighted bridging connections across layers over a mini-batch. GradMax uses left singular vectors of this auxiliary gradient as fan-out weights to maximise the gradient norm, while NeST initializes a sparse neuron connecting neuron pairs with a high gradient value in their auxiliary connection.

No known works add new neurons that are explicitly more different to existing neurons than randomly selected weights. In the context of non-growing and linear networks, [251] proposes orthogonal weight initialization. In the case of MLPs with nonlinear activation functions, a layer with orthogonal weights only provides orthogonal pre-activations, and only if the output of the previous layer is orthogonal. [252] finds pre-activation orthogonality may actually be detrimental in non-linear networks and presents a weight initialization scheme for layers with equal static widths that maximises post-activation orthogonality, but does not have a clear extension to neurogenesis. [133] proposes a NAS heuristic based on post-activation distances between samples, computationally similar to the orthogonality gap metric by [252]. Maintaining orthogonality of neuron activations in static networks has especially been studied in reinforcement learning [253] and self-supervised learning [254], but becomes relevant in all tasks when considering dynamically growing networks.

Neurogenesis during the learning process necessarily expands the parameter search space and thus the loss landscape's dimensionality. Neurogenesis may either immediately preserve the ANN's function, such as by using zero fan-in or fan-out weights as in [142, 152], or may change the function and current performance. Function-preserving neurogenesis is a form of network morphism [137, 255],

Algorithm 4 Neurogenesis framework.

procedure NEUROGENESIS(TRIGGER, INITIALIZATION, initial ANN f)
  while f not converged **do**
    Gradient descent step on current existing weights
    for each hidden layer l **do**
      if TRIGGER$(f, l) > 0$ **then**
        Add TRIGGER(f, l) neurons using INITIALIZATION(f, l)
  return trained f

Table 7: Specification of initialization and trigger pairs used for each strategy.

Strategy	Trigger		Initialization	
NORTH-Select	T_{act}	eq. (28)	Select	sec. 6.2.2
NORTH-Pre	T_{act}	eq. (28)	Pre-activation	eq. (33)
NORTH-Random	T_{act}	eq. (28)	RandomInit	sec. 6.2.2
NORTH-Weight	T_{weight}	eq. (31)	Weight	eq. (34)
GradMax	T_{grad}	eq. (32)	GradMax	sec. 6.2.2, [142]
Firefly	T_{grad}	eq. (32)	Firefly	sec. 6.2.2, [110]
NeST	T_{grad}	eq. (32)	NeST	sec. 6.2.2, [68]

maintaining the location in the newly expanded loss landscape but changing the direction of the descent path into the new dimensions. [142] selects the steepest direction by maximising the gradient norm. Function-altering neurons jump from the descent path: [110, 250, 256] escape convergence at saddle points, while [68] performs a backpropagation-like jump.

Previous neurogenesis works take many approaches across a range of motivations and applications to tackle this difficult problem. In order to unify neurogenesis strategies and study their components, we propose a framework within which we formulate our contributions.

6.2 A Framework for Studying Neurogenesis

Our neurogenesis framework decomposes neurogenesis strategies into *triggers*, which are heuristics that determine when, where, and how many neurons to add, and *initializations*, which determine how to set their weights before training them. We present the basic framework in Algorithm 4. After every gradient step, for each layer, the trigger is evaluated to compute if and how many neurons to add, then the initialization is used to add these new neurons. As the trigger is evaluated after each gradient step, the decision of when is based on non-zero outputs from the trigger. Similarly, the question of where to add neurons is treated by evaluating the trigger on each layer independently.

We introduce our contributions for the case of MLPs, although they generalize to more complex cases like convolutions, detailed in Section 6.2.3. Our strategies with their triggers and initializations are summarized in Table 7. We define triggers in Section 6.2.1 and initializations in Section 6.2.2.

6.2.1 Triggers

We explore three sources of information for triggers for neurogenesis: neural activations, weights, and gradients. These triggers determine if and how many neurons to add for each layer after each gradient step. We propose that a useful neurogenesis trigger should add neurons when doing so will efficiently improve the capacity of the network or learning.

Activation-based When building an efficient network through neurogenesis, we intuitively want to add neurons yielding novel features that may improve the direction of descent, lower asymptotical empirical risk, or avoid unnecessary redundancy in the network. Thus, we need to measure how different or orthogonal the post-activations are from each other. To measure orthogonality of a layer, we use the ϵ-numerical rank of the post-activation matrix [253, 257]. For layer l and n samples generating the post-activation $\boldsymbol{H}_l$, the effective dimension metric may be estimated by

$$\phi_a^{ED}(f,l) = \frac{1}{M_l}\left|\left\{\sigma \in \text{SVD}\left(\frac{1}{\sqrt{n}}\boldsymbol{H}_l\right) \middle| \sigma > \epsilon\right\}\right|, \tag{27}$$

where $\text{SVD}\left(\frac{1}{\sqrt{n}}\boldsymbol{H}_l\right)$ is the set of singular values of $\frac{1}{\sqrt{n}}\boldsymbol{H}_l$ and $\epsilon > 0$ is a small threshold. Due to the SVD decomposition, evaluating this metric has a constraint on the dimensions of $\boldsymbol{H}_l$ of $n > M_l$.

When the orthogonality metric of a layer's current post-activations is high, there may be additional useful features beyond the currently saturated directions so we add neurons. Conversely, a low metric value indicates redundancy in the layer, which may benefit from differentiation between neurons via gradient steps before reconsidering neurogenesis. Adding a new neuron will increase the metric when its activation is more orthogonal to that of other neurons and decrease it if it is redundant. We use each layer's metric value at network initialization as the baseline value, which accounts for orthogonality loss as the input is propagated through layers the network. We aim to at least maintain this initial metric value over training, even as the layer grows: if new neurons do not increase the rank and thus lower the metric value, then neurogenesis pauses until the rank increases via gradient descent. We multiply the baseline value by a threshold hyperparameter γ_a close to 1. The number of neurons triggered is the difference of the current metric value and the threshold, scaled by the current number of neurons:

$$T_{act}(f,\phi_a,l) = \min\left(0, \lfloor M_l\left(\phi_a(f,l) - \gamma_a\phi_a(f_0,l)\right)\rfloor\right), \tag{28}$$

where f_0 is the ANN at initialization and ϕ_a is the orthogonality metric.

We additionally study a metric based on the orthogonality gap from [252] that measures the gap of the covariance matrix of post-activations to the identity matrix. We use an estimate of the orthogonality gap

$$\phi_a^{OG}(f,l) = 1 - \left\|\left(\frac{1}{\|\boldsymbol{H}_l\|_F^2}\right)\boldsymbol{H}_l^\intercal\boldsymbol{H}_l - \left(\frac{1}{\|\boldsymbol{I}_n\|_F^2}\right)\right\|_F, \tag{29}$$

where $\boldsymbol{I_n} \in \mathbb{R}^{n\times n}$ is the identity matrix. $\boldsymbol{H}_l^\intercal\boldsymbol{H}_l \in \mathbb{R}^{n\times n}$ represents the covariance of $\boldsymbol{H}_l$ across samples and thus approaches the identity matrix if the post-activation vector of each neuron is orthogonal to all others. However, this metric has a dependency on layer width that may confound neurogenesis strategies. Also, this metric does not extend to the weight-based strategy presented next.

Weight-based We include weight matrix orthogonality as a comparison to activation-based methods. The trigger, T_{weight}, is computed as in equations (27)-(28) but with $\boldsymbol{W}_l$ instead of $\boldsymbol{H}_l$:

$$\phi_w^{ED}(f,l) = \frac{1}{M_l}\left|\left\{\sigma \in \text{SVD}\left(\frac{1}{\sqrt{n}}\boldsymbol{W}_l\right) \middle| \sigma > \epsilon\right\}\right|, \tag{30}$$

$$T_{weight}(f,\phi_w,l) = \min\left(0, \lfloor M_l\left(\phi_w(f,l) - \gamma_w\phi_w(f_0,l)\right)\rfloor\right). \tag{31}$$

Gradient-based In order to study gradient-based neurogenesis initializations such as [68], [110], and [142] in the context of dynamic neurogenesis, we propose a trigger based on the auxiliary gradient. As shown by [142], the maximum increase in gradient norm by adding k neurons to the l^{th} layer is the

sum of the largest k singular values of the auxiliary gradient matrix $\frac{\partial L}{\partial \boldsymbol{z}_{l+1}} \boldsymbol{h}_{l-1}^{\intercal}$. We use these singular values and compare them to the gradient norms of all existing neurons in that layer over the same n samples. The number of neurons triggered is the number of singular values larger than the sum of gradient norms:

$$T_{grad}\left(f, L, l\right) = \left| \left\{ \sigma \in \text{SVD}\left(\frac{\partial L}{\partial \boldsymbol{Z}_{l+1}} \boldsymbol{H}_{l-1}^{\intercal} \right) \middle| \sigma > \sum_{m=1}^{M_l} \left\| \frac{\partial L}{\partial \boldsymbol{w}_m^{\text{in}}} \right\|_F + \left\| \frac{\partial L}{\partial \boldsymbol{w}_m^{\text{out}}} \right\|_F \right\} \right|, \tag{32}$$

where $\boldsymbol{w}_m^{\text{in}}$ is the m^{th} row of $\boldsymbol{W}_l$ and $\boldsymbol{w}_m^{\text{out}}$ is the m^{th} column of $\boldsymbol{W}_{l+1}$, respectively representing the fan-in and fan-out weights of the m^{th} neuron in layer l. We intuit this threshold creates neurons that could have initial gradients significantly stronger than existing neurons and will be harder to surpass as the layer grows, thus leading to convergence in layer width.

6.2.2 Initializations

For the initialization of new neurons, we aim to reuse information computed for the corresponding trigger in each method to reduce computational overhead. For all NORTH* initializations, we add function-preserving neurons which do not immediately change the output of the network, or more specifically the activation of any downstream layers. We propose that the role of initialization for neurogenesis during training is rather to add a new neuron that is useful locally to the triggered layer which will then be integrated into the network's functionality through gradient descent; we measure this utility based on activation, weights, and gradients. We rescale all new neurons to match the weight norm of existing neurons, so scaling is ignored in the following calculations for simplicity.

Activation-based We present multiple approaches for initializing neurons towards orthogonal post-activations. Due to the nonlinearity of a neuron, we only have a closed-form solution for a set of fan-in weights yielding not the desired orthogonal post-activation but an orthogonal pre-activation, similar to related works on orthogonality in static networks [251]. The following approaches generate candidate neurons and select those that independently maximise the orthogonality metric of the post-activations with the existing neurons in that layer. We initialize fan-out weights to 0.

- Select: generate random candidate neurons.
- Pre-activation: generate candidate neurons with pre-activations maximally orthogonal to existing pre-activations. For n samples and M_l neurons in layer l, a candidate's fan-in weights are

$$\boldsymbol{w} = (\boldsymbol{H}_{l-1}^{\intercal})^{-1} \boldsymbol{V}_{\boldsymbol{Z}_l}'^{\intercal} \boldsymbol{a}, \tag{33}$$

  where $(\boldsymbol{H}_{l-1}^{\intercal})^{-1}$ is the left inverse of the transpose of post-activations $\boldsymbol{H}_{l-1}$ for layer $l-1$, $\boldsymbol{V}'_{\boldsymbol{Z}_l}$ is comprised of orthogonal vectors of the kernel of pre-activations $\boldsymbol{Z}_l$, and $\boldsymbol{a} \in \mathbb{R}^{n-M_l}$ is a random vector resampled for each candidate.

As a baseline to these selection-based techniques, NORTH-Random uses RandomInit as the initialization strategy with random fan-in weights and zeroed fan-out weights.

For pre-activation based initialization, we compute the Moore-Penrose pseudoinverse $(\boldsymbol{H}_{l-1}^{\intercal})^{+}$, which approximates the left inverse when $n \gg M_{l-i}$ and the rows of $\boldsymbol{H}_{l-1}$ are approximately orthogonal. The orthogonal vectors of the kernel of pre-activations $\boldsymbol{Z}_l$ are computed as the right $n - M_l$ columns of $\boldsymbol{V}_{\boldsymbol{Z}_l}$ from the full singular value decomposition of $\boldsymbol{Z}_l$.

In addition to the two methods presented for creating candidates for activation-based selection at neuron initialization, we also present an optimization-based approach. We perform projected gradient

descent [258] on randomly generated candidate neurons, optimizing the orthogonality metric of each set with the existing neurons and constrained by weight norm.

When the effective dimension metric is used, we need a continuous proxy for selection across all three methods as well as a differentiable proxy for the projected optimization. We use the sum of singular values as the differentiable proxy as well as the tie-breaker for neurons that yield equivalent effective dimensionalities for the layer.

However, the optimization method was found to be prohibitively expensive, performing an inner optimization for every growth event with an additional projection step for every gradient step. Further study into post-activation orthogonality metrics and proxies is warranted, particularly balancing performance with efficiency.

Weight-based For the weight-based strategy NORTH-Weight, maximally orthogonal weights for new neurons can be solved for directly. We initialize k neurons each with fan-out weights of 0 and fan-in weights projected onto the kernel of $\boldsymbol{W}_l$:

$$\boldsymbol{w} = \text{proj}_{\ker W_l}(\boldsymbol{w}_i), \tag{34}$$

where $\boldsymbol{w}_i \in \mathbb{R}^{M_{l-1}+1}$ are the random initial fan-in weights from the base weight initialization and the projection is computed using $\boldsymbol{V}'_{\boldsymbol{W}_l}$, comprised of orthogonal vectors of the kernel.

Gradient-based We reimplement the neurogenesis initialization portions of NeST from [68], Firefly from [110], and GradMax from [142] within our framework to investigate them in the dynamic context and be able to compare them with our dynamic methods.

- NeST from [68]: initialize k neurons, using sums of squares of the largest auxiliary gradient matrix values with a random sign to the respective fan-in and fan-out weights as connections. As NeST is only specified for a single neuron, we adapt to k neurons by using different random signs for each neuron. We use the same recommended top quantile of 0.4 for filtering the largest auxiliary gradient matrix values. In the convolutional case, generate random candidate neurons and add those that independently lower the loss the most.
- Firefly from [110]: generate candidate neurons from two strategies: split existing neurons by halving the fan-out weights and adding and subtracting random uniform noise from the two copies, and add new candidates with random fan-in weights and small fan-out weights. We then keep the k neurons with the largest gradient norm. We do not perform gradient steps on the candidates, as done in Firefly, in order to focus on neurogenesis rather than pruning after training; Firefly does both. The magnitude of the noise for split candidates and the fan-out weights for new candidates is controlled by hyperparameter $\epsilon_{\text{Firefly}}$, for which we use a value of 1e-4.
- GradMax from [142]: initialize k neurons with zero fan-in weights and the top k left singular vectors of the auxiliary gradient matrix as the fan-out weights. We note that this method is limited to M_{l+1} candidate neurons, capping neurogenesis at the size of the next layer.

6.2.3 Neurogenesis for Convolutions

For convolutional layers, we consider that a channel is analogous to a neuron in a dense layer. However, the activation for a channel and a single sample as well as the parameterization of the connection between two channels are both matrices instead of single values. Thus, $\boldsymbol{H}_l \in \mathbb{R}^{H_l \times W_l \times M_l \times n}$ and $\boldsymbol{W}_l \in \mathbb{R}^{k_l \times k_l \times M_l \times (M_{l-1}+1)}$. We detail how our methods are adapted to convolutions as follows.

- Effective Dimension: $\boldsymbol{H}_l$ is flattened to $\mathbb{R}^{M_l \times (H_l W_l n)}$, so that the metric is relative to the number of neurons in layer l. This permits the use of a smaller buffer size.
- Orthogonality Gap: $\boldsymbol{H}_l$ is flattened to $\mathbb{R}^{(H_l W_l M_l) \times n}$, so that $\boldsymbol{H}_l^\intercal \boldsymbol{H}_l \in \mathbb{R}^{n \times n}$.
- Activation-based Trigger: Because the orthogonality metric at initialization is very close to 0 for the last layers of a deep convolutional network and increases as the network learns, we instead use the running maximum of orthogonality metric values for the threshold. Thus,

$$T_{act}^{conv}(f, \phi_a, l) = \min\left(0, \left\lfloor M_l\left(\phi_a(f,l) - \gamma_a \max_t \phi_a(f_t, l)\right)\right\rfloor\right). \tag{35}$$

  In the case of residual networks, we turn off residual connections when evaluating activation-based metrics to isolate the contribution by each non-identity layer. We also grow groups of layers with interdependent channel constraints due to these residual connects based on metrics of the first layer of the group.
- Weight-based Trigger and Initialization: $\boldsymbol{W}_l$ is flattened to $\mathbb{R}^{M_l \times ((M_{l-1}+1)k_l k_l)}$.
- Gradient-based Trigger: While we did not have enough resources to include results for all dynamic gradient-based neurogenesis strategies in convolutional neural networks, the implementation for these are included in our code. Computing the auxiliary gradient is detailed in [142].
- Activation-based Initialization: We do not have a closed-form solution for orthogonal pre-activations of convolutions, so we do not implement NORTH-Pre for architectures with convolutions.

6.3 Experiments

We study the impact of the different trigger and initialization methods detailed in Table 7 over a variety of tasks, with dynamic schedules to study triggers and independently studying the importance of initialization by also using fixed schedules. We also compare with static networks of various sizes to understand the relative performance of networks grown with neurogenesis. Specifically, we focus on three settings: MLPs [259] on generated toy datasets in Section 6.3.2, MLPs on MNIST [260] in Section 6.3.3, and convolutional neural networks (CNNs) on image classification: VGG-11 [261] and WideResNet-28 [262] on CIFAR10/CIFAR100 in Section 6.3.4. For all experiments, plots show mean and standard deviation in the form of error bars, clouds, or ellipses (which may result in artifacts outside of the layer size bounds) or quartiles in the form of boxplots across 5 seeded trials. We include further experiment details in Section 6.3.1 and a study on hyperparameters in 6.3.5. We only compare our results within our framework rather than take reported results in order to avoid confounding differences in implementation and focus on understanding neurogenesis. By studying neurogenesis across different tasks, architectures, and layer types, we aim to draw general conclusions about when, where, and how to add new neurons to ANNs.

6.3.1 Experiment Details

We run 5 trials with random seeds 1-5 in each study for each configuration and present aggregated results. We list hyperparameters in Table 8. For initializations that generate candidates, the number of candidates is set to either 1000 for dense layers or 100 for convolutional layers, plus the number of neurons to add. For all initializations, the bias is initially set to 0 and non-zero fan-in weights and fan-out weights of new neurons are scaled to match the current weight norm of existing neurons in the same layer for all initializations, except for Firefly for which we scale noise vectors to $\epsilon_{\text{Firefly}}$ = 1e-4 times the current weight norm of existing neurons. We use a modified ReLU activation function as GradMax

Table 8: Default hyperparameters.

	MLP - Generated	MLP - MNIST	VGG-11 - CIFAR10	VGG-11 - CIFAR100	WRN-28 - CIFAR10
Optimizer	ADAM [263]		ADAM, CosineAnnealing		
Learning Rate	3e-4				3e-3
Batchsize	128	512	128		
Epochs	Convergence	20	100		50
Input dimensions	64	784	$32 \times 32 \times 3$		
Output dimension	2	10	10	100	10
Hidden layers/groups	1 or 2	2	10		4
Initial layer width	4	64	0.25×		
Medium Static width	N/A	256	1×		
Maximum layer width	512	784	2×		6×
Buffer size	1024	1568	128		
Base initialization	Xavier uniform [264]				
Orthogonality metric	ϕ_{ED}				
ϵ, eq. (27),(30)	0.01				
γ_a, eq. (28)	0.97		0.9,0.97,0.99	0.9	
γ_w, eq. (31)	0.99				
Device	CPU		GPU		

requires that the gradient at 0 is 1; in other words, $\sigma(x) = 0$ if $x < 0$ and $\sigma(x) = x$ if $x \geq 0$, resulting in the same activation but a different gradient at 0 compared with standard ReLU $\sigma(x) = \max(0, x)$. We noted no empirical difference in results between the standard and modified ReLU and thus use the modified ReLU across all configurations [265].

We compare growing networks against static networks: Small Static is the same size as the initial size of growing networks, Medium Static is the same size as the final size for preset schedules and is also the baseline used for preset schedules, and Big Static is the same size as the maximum size of growing networks.

All plots except 32 average across the 5 random seeds and use standard deviation for error bars, clouds, or ellipses. The ellipses represent the contour line of a Gaussian density function matching the mean and covariance at 1 standard deviation.

We ran all CPU experiments on Intel Xeon Gold 6136 computing nodes and the GPU experiments on GTX 1080TIs, RTX8000s, and Tesla V100s, all provided by local clusters. The experiments to generate all plots and tables in this work consumed 299.53 CPU days and 85.17 GPU days in total, with an estimated carbon footprint of 236.3 kgCO_2e [266, 267].

6.3.2 Generated Data

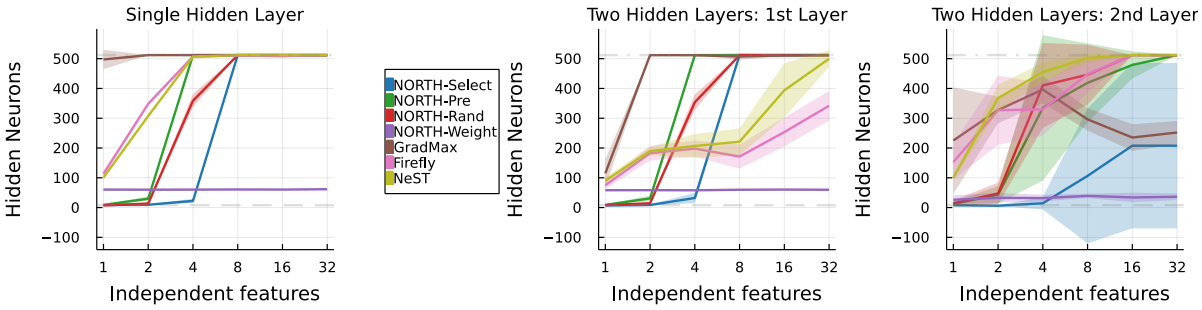


Figure 28: Hidden neurons upon convergence on for MLPs on generated toy data versus the number of independent features in a 1 (left) and 2 (middle, right) hidden layer MLPs.

We first analyze all strategies on supervised learning for toy generated binary classification datasets. All have 64 input features but differ in how many of these inputs are linearly independent, demonstrating how neurogenesis responds to the effective dimensionality of the input. The datasets are created for

a given number of independent features n_f by generating 5000 samples of n_f normally distributed features and filling the remaining $64 - n_f$ features with random linear combinations. The output binary class for each sample is generated by summing all features and adding 10% random noise, then thresholding by the mean value. 10% of samples are reserved as the test split for final accuracy. We train MLPs with 1 or 2 hidden layers using the different neurogenesis strategies in all layers, shown in Figure 28. In this simple task, all strategies achieve similarly high test accuracy across datasets: above 99% for all feature sizes except 97% for 32 independent features.

Comparing final layer widths shows the dependency of each strategy on the effective input dimensionality and base architecture. The first layers across the two architectures exhibit the unsupervised nature of NORTH* strategies but also the strength of gradient-based strategies to adapt width to depth. Width of the second hidden layer has higher variance across all strategies, indicative of the layer-wise input distribution: the input to the second layer is changing in dimension and distribution, whereas the first hidden layer has a static input from the dataset in this task.

NORTH-Weight is restricted by the input dimensionality due to the upper bound on number of singular values by the minimum of the matrices' dimensions. Thus, it cannot explore networks wider than the input dimensionality, which may be problematic for tasks with low-dimensional inputs. All other NORTH* methods grow to the maximum size allowed in the first layer and to larger sizes than NORTH-Weight in the second layer.

6.3.3 MLP MNIST

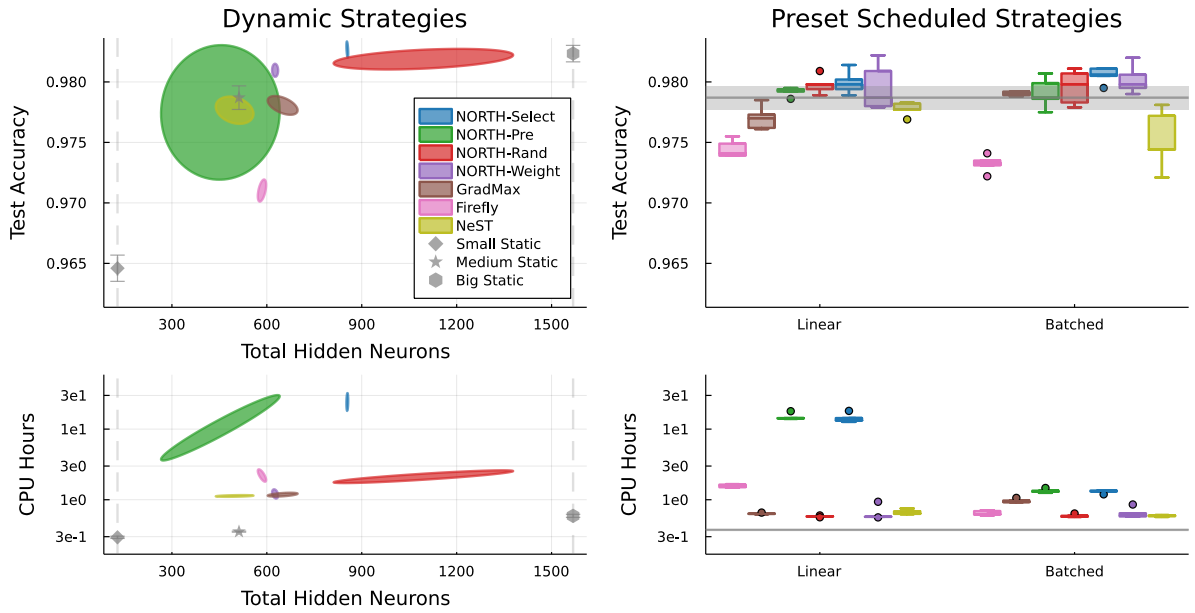


Figure 29: Dynamic (left) and preset scheduled (right) comparisons of network performance across neurogenesis strategies for MLPs on MNIST, evaluated by test accuracy (top) and training time (bottom). Preset scheduled are compared against the Medium Static network, which is the same architecture as the final size of the grown networks.

We continue our study on neurogenesis in dense MLP layers on the task of MNIST classification. The 28×28 images of the MNIST dataset are flattened to 784 features. This is fed into an MLP with 2 hidden layers which grow using neurogenesis, then classified into 10 classes representing the digit in the image. We use the standard training and test split.

We test dynamic strategies as well as isolate the initializations in preset growth schedules. To better understand the contribution of the trigger versus initialization, we compare initialization methods using fixed neurogenesis schedules. We use two predetermined schedules: Linear, which adds one

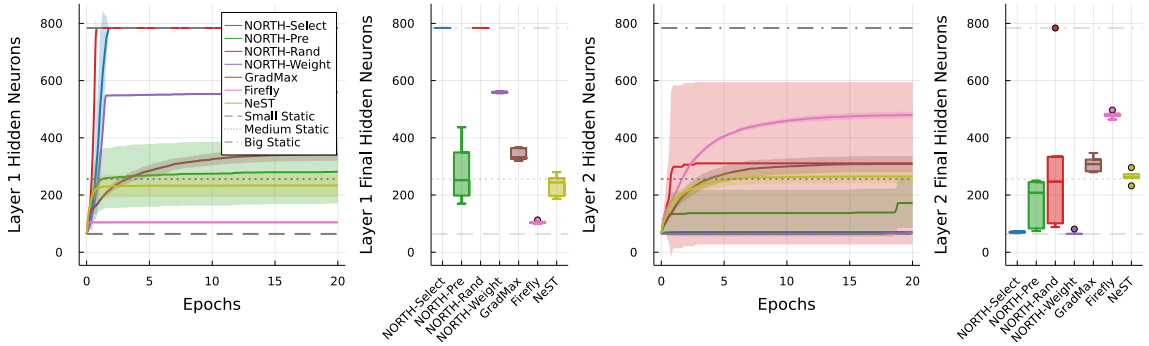


Figure 30: Layer width for MLPs on MNIST over training for dynamic strategies, across training (far-left for Layer 1, center-right for Layer 2) and at the end of training (center-left for Layer 1, far-right for Layer 2).

neuron per gradient step, and Batched, which adds batches of neurons after each gradient step. Both schedules are defined by initial and final layer widths. We grow neurons for the first 75% of epochs. The linear schedule adds neurons one at a time so it has many small growth events, while the batched schedule adds neurons in 8 regular intervals so it has only a few large growth events. We compare with a Medium Static network which starts training with the final size of the growing networks, 256 neurons per hidden layer. We compare test accuracy and training time against final network size or schedule in Figure 29, detailing the layer widths over training for dynamic schedules in Figure 30.

For dynamic neurogenesis, the Pareto front of accuracy versus network size is dominated by the NORTH* orthogonality-based methods, demonstrating the advantage of using activation and weight orthogonality to trigger neurogenesis.

The gradient-based methods, especially GradMax and NeST, yield comparable but not quite competitive results, possibly confounded by the adaptations used to implement them within our framework. However, they are faster in this CPU implementation than some activation-based NORTH* strategies, benefiting from thorough code optimization of deep learning libraries for gradient descent, while the larger SVD calculations used by NORTH* strategies hinder efficiency.

From Figure 30, no dynamic strategy reached the maximum possible network size. Thus, our triggers do respond to the "When" and "How many" questions, converging towards smaller networks. Furthermore, NORTH-Select and NORTH-Random reach networks which are competitive with the largest non-growing network with fewer neurons. NORTH* methods tend to converge towards networks with large first layers and smaller second layers, a common pattern in hand-designed networks. Thus, the triggers can also respond to the "Where" question by dynamically adding neurons to each layer independently. However, using a maximum layer size is still needed for these methods to avoid exploding layer widths, particularly without extensive hyperparameter tuning.

As for preset schedules, multiple NORTH* strategies exceed the performance of the static network of the same final architecture for this task and hyperparameter setting, demonstrating that dynamically growing a network over learning can result in better performance than starting with a network of the maximum final size. NORTH-Pre slightly under-performed the less-informed approaches of NORTH-Select and NORTH-Random, as well as NORTH-Weight which has a similar premise towards orthogonal pre-activations.

6.3.4 Deep CNNs on CIFAR10/CIFAR100

We extend neurogenesis to deep convolutional networks and test them on CIFAR10 and CIFAR100 classification with a VGG-11 or WideResNet-28 backbone. We use the standard training and test splits of CIFAR10 and CIFAR100, which have input images of size $32 \times 32 \times 3$ and 10 or 100 classes, respectively. The standard VGG-11 hidden layer widths are 64, 128, 256, 256, 512, 512, 512, and 512 channels in the

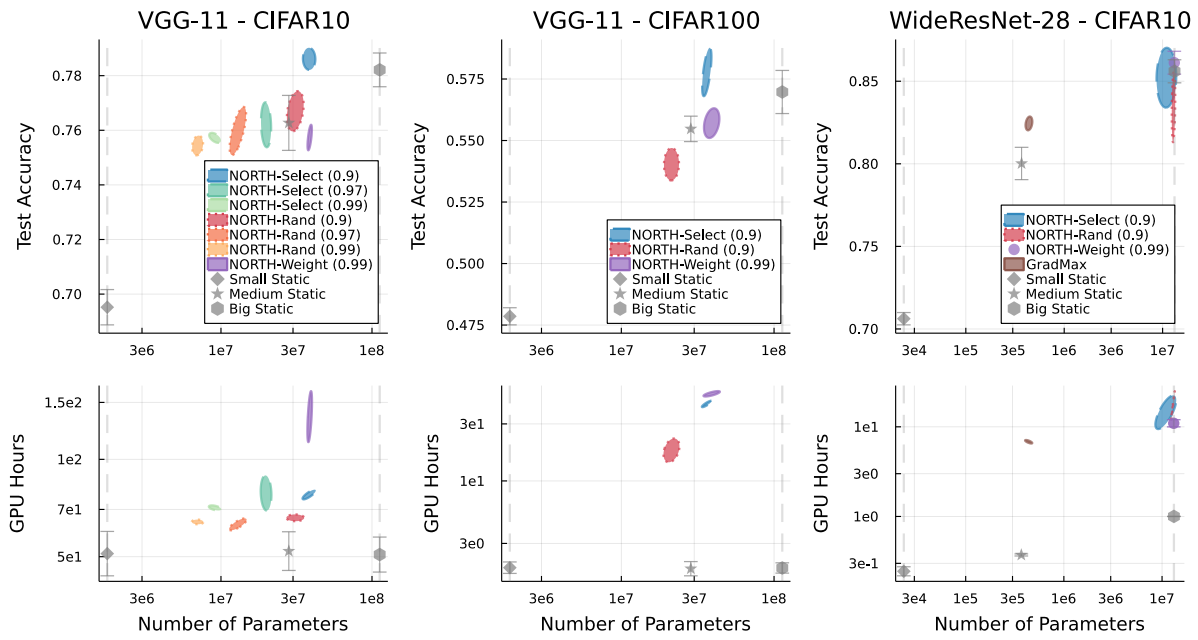
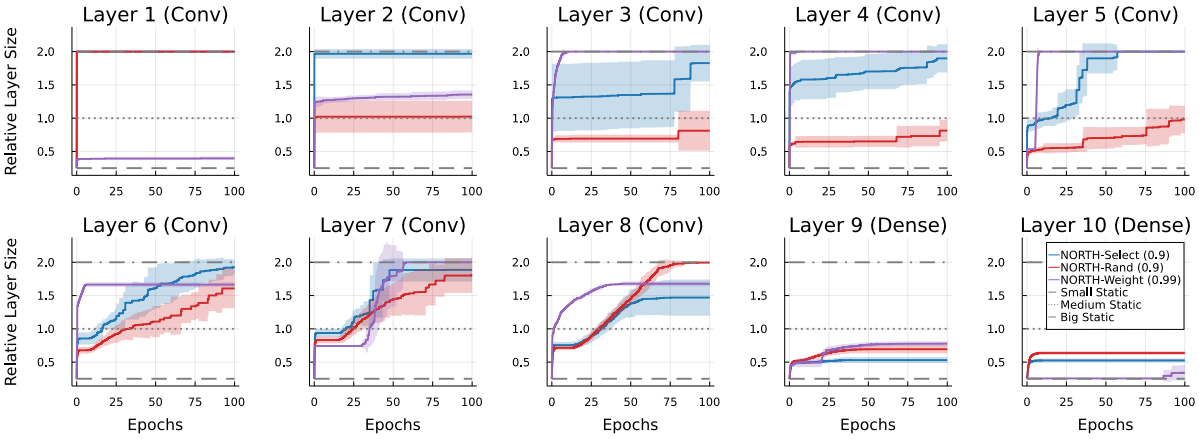


Figure 31: Comparisons of test accuracy (top) and training time (bottom) vs. network size across neurogenesis strategies and values of γ for VGG-11 on CIFAR10 (left) and CIFAR100 (center) and WideResNet-28 on CIFAR10 (right).

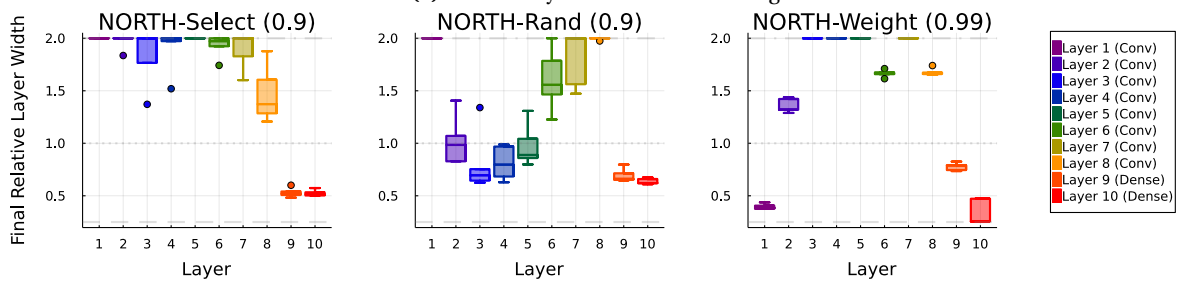
Table 9: Average GPU seconds per mini-batch for trigger evaluation for all layers, including GPU memory garbage collection time, during the 1st epoch of VGG-11 training on CIFAR10 on a Tesla V100.

Strategy	Time
NORTH-Select	5.97 ± 0.18
NORTH-Rand	5.77 ± 0.09
NORTH-Weight	10.05 ± 1.01
GradMax	14.00 ± 7.03
Firefly	18.52 ± 11.59
NeST	23.79 ± 8.78
Small Static	5.49 ± 0.05
Medium Static	5.49 ± 0.05
Big Static	5.14 ± 0.05

convolutional layers, then 4096 and 4096 neurons in the dense layers. The standard WideResNet-28-1 has an initial convolution layer with 16 channels, 3 groups with 16, 32, and 64 channels each and 4 residual blocks per group, and a final dense layer connecting global mean pooling to the output. The backbone used as the starting point for neurogenesis has $\frac{1}{4}\times$ the standard layer widths, also used for the Small Static architecture. The layer sizes for growing networks were limited to $2\times$ the standard layer widths for VGG-11 and $6\times$ the standard layer widths for WideResNet-28-1 (which is WideResNet-28-6 in their notation), also used for the Big Static architecture. hlDue to the interdependence of layer widths in WideResNet-28, we measure the first layer of each group to determine when, where, and how to add neurons to all layers of the same group. We also turn off the residual connections when evaluating activation and weight-based metrics. We use batch-normalization in WideResNet-28, but otherwise do not perform any other "tricks" such as data augmentation or dropout. Due to restricted computational time, we could not perform a formal sweep of hyperparameters. We selected the largest learning rate that we tested that allowed all networks up to Big Static to increase accuracy during the first 5 epochs. Similarly, we selected the largest batchsize that we tested that could fit any gradient computation in memory on all GPU devices. The cluster GPUs available to us were not identical, so we ran each base architecture and dataset combination on the same GPU in order to compare within



(a) Relative layer widths over training.



(b) Relative layer widths at the end of training.

Figure 32: Layer widths for VGG-11 on CIFAR10, scaled relative to standard VGG-11 widths, for NORTH-Select, NORTH-Rand, and NORTH-Weight, over the course of training in (a) and at the end of training in (b).

experiments, but not across.

The gradient-based strategies require computing auxiliary gradients in addition to the gradients of existing parameters and any gradients further required in gradient-based initialization, shown in Table 9. Because batchsizes are generally selected to maximise GPU memory of each gradient evaluation and the gradient-based neurogenesis methods use multiple passes per batch, they are less efficient than NORTH* in the case of VGG-11.

Test accuracy and training cost of the different methods is compared against network size for each base architecture and dataset combination in Figure 31. On VGG-11, our NORTH* methods find networks in the same order of size of the standard network, here represented by Medium Static, while being competitive or improved in accuracy for both datasets. On CIFAR10, we search across γ_a, showing the tunability of NORTH-Select and NORTH-Rand across the trade-off for size and performance. Notably, NORTH-Select with $\gamma_a = 0.9$ outperforms even Big Static with less than half as many parameters on both CIFAR10 and CIFAR100. As in the dense case, NORTH-Select has a slightly higher cost for the benefit of smarter candidate generation compared to NORTH-Rand. For WideResNet-28, NORTH* methods tended to grow to the at or near the maximum size, likely due to the complicated dynamics of residual connections on the trigger metrics. NORTH-Weight slightly beats Big Static at this maximum size while NORTH-Select matches performance, and GradMax beats Medium Static at a similar size.

Layer width during and at the end of training for VGG-11 on CIFAR10 is presented in Figure 32, showing how different NORTH* methods respond to the "Where" question. All three methods keep the dense layers relatively small, which has a large effect on the overall parameter count. NORTH-Weight grows some intermediate convolutional layers to maximum size, almost opposite to the width patterns of NORTH-Rand. Deeper convolutional layers tend to have later innovations, likely in response to innovations of the earlier layers.

6.3.5 Hyperparameter studies on MLPs for MNIST

In the following experiments, we study the effect of NORTH* hyperparameters independently in the MNIST task. These experiments were used to inform our default settings for the experiments in 6.3.3. We discuss our findings in the context of this specific task and setup. We only use the standard training split of MNIST in these experiments, holding out 10% of this split as the validation set to measure final validation accuracy.

We study the effects of activation orthogonality measures Effective Dimension ϕ_a^{ED} from Equation 27 versus Orthogonality Gap ϕ_a^{OG} from Equation 29 on performance and training time and their sensitivities to γ_a in Figure 33. ϕ_a^{ED} generally yields higher accuracies given the network size, although ϕ_a^{OG} is generally more efficient. ϕ_a^{ED} is more robust to varying γ_a and yields less extreme network sizes, although both metrics are rather sensitive to its value. Thus, applying activation-based neurogenesis to new tasks may require tuning of γ_a to the specific task configuration. ϕ_a^{ED} has the additional hard constraint on sample size being greater than the current layer width, which may make scaling to very wide networks difficult.

We study the effect of batchsize in Figure 34a. We note that the size of the buffer used for assessing the orthogonality metric is independent of batch size. The dynamic networks generally benefit more from larger batchsizes than static networks. NORTH-Select particularly matches or exceeds the validation performance of larger static networks at larger batchsizes. We hypothesize that the relative frequency of neurogenesis trigger and initialization steps for larger batchsizes is beneficial.

The results of various learning rates are shown in Figure 34b. The general trend is that larger networks, both static and growing, benefit from smaller learning rates. NORTH-Select matches NORTH-Random in validation performance at smaller learning rates, even with a smaller network size. We conjecture that larger learning rates cause large, noisy gradient steps that hinder the activation-based triggering of neurogenesis while increasing randomness and thus orthogonality for weight-based triggering.

6.4 Discussion

NORTH* strategies grow efficient networks which achieve comparable performance to static architectures, even surpassing the performance of static architectures of the same final size. We posit that the use of growth with informed triggers leads to efficient networks as redundant neurons have a low chance of being created in the small network initialization and are not added by NORTH* neurogenesis. This highlights a benefit of neurogenesis: dynamically creating an architecture adapted to the target task.

We find that activation and weight orthogonality can be useful triggers for dynamic neurogenesis and weight initializations of new neurons. Previous methods use gradient information for these decisions, explicitly considering training dynamics and particularly the outputs via the loss. Our NORTH* methods only implicitly use this information via network updates changing orthogonality metrics and explicitly use the input distribution. This could allow extension to unsupervised and semi-supervised contexts, where gradient information may be unavailable or unreliable.

As neurogenesis triggers are evaluated after every gradient step, they can have significant computational cost. NORTH* strategies were more efficient than gradient-based strategies in our GPU experiments, but less for CPU. The number of candidates, frequency of trigger evaluations, and the size of the activation buffer are hyperparameters of NORTH* which can be reduced for efficiency or increased for performance. Comparing the cost of growing networks to that of static networks is difficult: although dynamically grown networks incur higher training costs than predetermined schedules or non-growing networks, they have the benefit of algorithmically determining the architecture widths instead of hand-engineering them through expensive trial-and-error.

We focus on function-preserving neuron addition. Such neurons do not immediately impact network output while still receiving gradient information to become functional through training. This can be easily achieved by zero-ing either the fan-in weights (as in NORTH*) or the fan-out weights (as in GradMax). Firefly and NeST, however, initialize functional neurons that change the position in the loss landscape; this could hinder performance by moving the network in an undesired direction, but could also complement gradient descent through neuron initialization. Optimizing the fan-out weights of NORTH* strategies to make helpful movements in the loss landscape is a direction for future work.

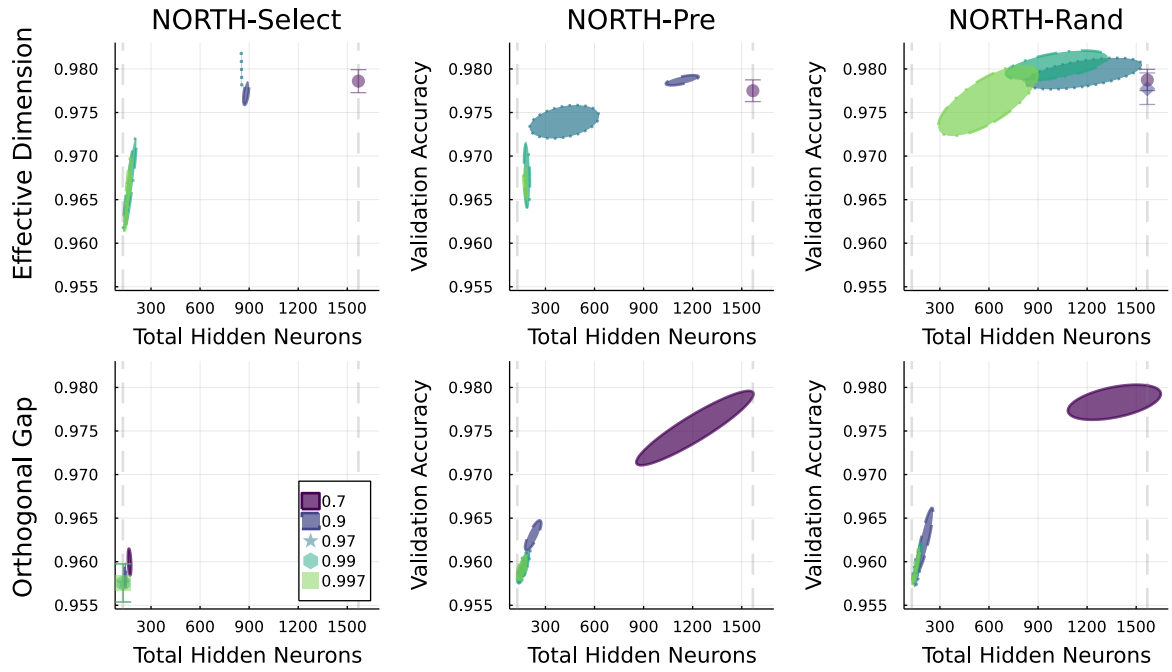
This work serves as a study to isolate neurogenesis from other structural learning techniques in order to understand it more and open the door to further neurogenesis research. We pair complimentary triggers and activations; other combinations are possible albeit with careful consideration. We show that neurogenesis alone can already find compact yet performant networks. We leave dynamic structural learning, incorporating neurogenesis along other structural operators like layer addition and pruning, as future work. For layer addition, we hypothesize that similar activation and weight-based triggering applies, then layer addition candidates could be compared to neuron addition candidates; for gradient-based methods, evaluating the auxiliary gradient on directly neighboring layers rather than alternating could signal layer addition. As for pruning, we hypothesize that unstructured pruning may not be necessary with our strategies that avoid redundancy at initialization, but structured pruning could remove neurons that become redundant over training. This could bypass the need for hyperparameters such as maximum layer width and be even more effective towards searching for networks that optimize both performance and cost.

Limitations and Broader Impact Statement: In addition to the limitations and broader impacts already discussed, the efficient and adaptive architectures grown by neurogenesis have reduced environmental footprint, avoiding cycles of hand-tuning static architectures or training expensive super-networks to prune or apply NAS to. We do not identify any potential negative societal impacts beyond those associated with general-purpose machine learning methodologies.

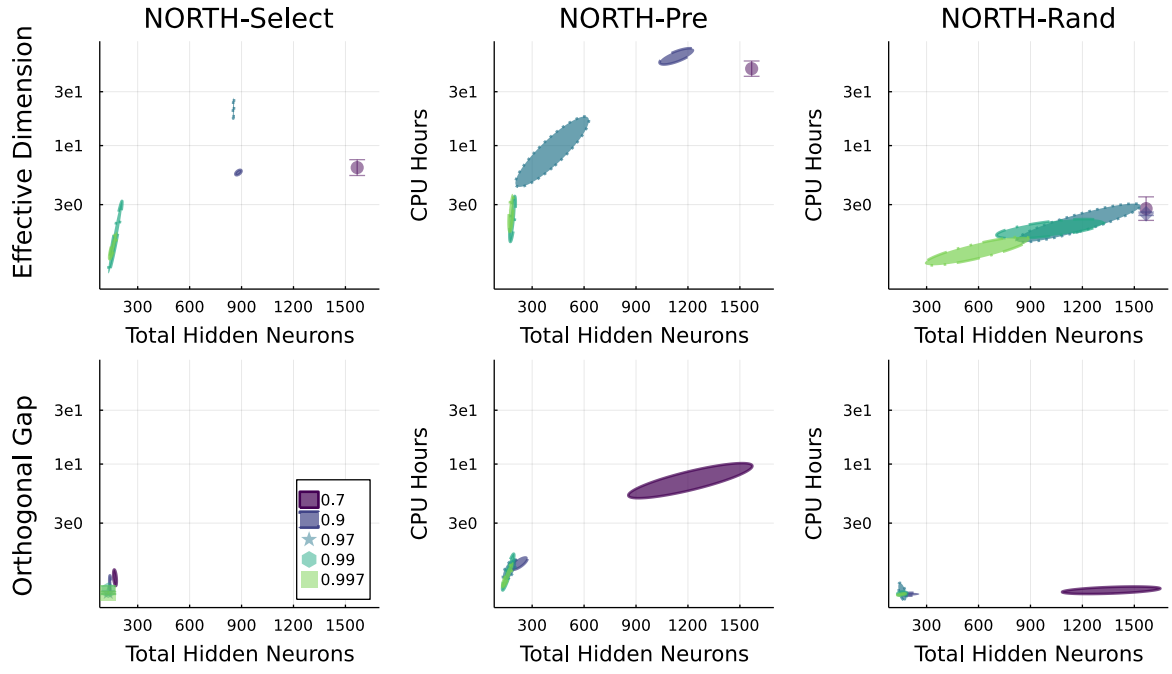
6.5 Conclusion

We present the NORTH* suite of neurogenesis strategies, comprised of triggers that use orthogonality metrics of either activations or weights to determine when, where, and how many neurons to add and informed initializations that optimize the metrics. NORTH* strategies achieve dynamic neurogenesis, growing effective networks that converge in size and can outperform baseline static networks in compactness or performance. The orthogonality-based neurogenesis methods in NORTH* could be further used to grow networks in a variety of settings, such as continual learning, shifting distributions, or combined synergistically with other structural learning methods. Neurogenesis can grow network architectures that dynamically respond to learning.

Understanding neurogenesis in ANNs permits its application as an operation in structural learning algorithms, particularly as a complement to pruning. This expands the potential search space used during structural learning, permitting more powerful and adaptive methods. Furthermore, the investigated dynamic scheduling aims to perform neural creation exactly when necessary and useful, removing the need for manually designed scheduling and increasing adaptivity. This study on neurogenesis is extended with neural pruning for dynamic learning environments such as transfer learning in the following section.

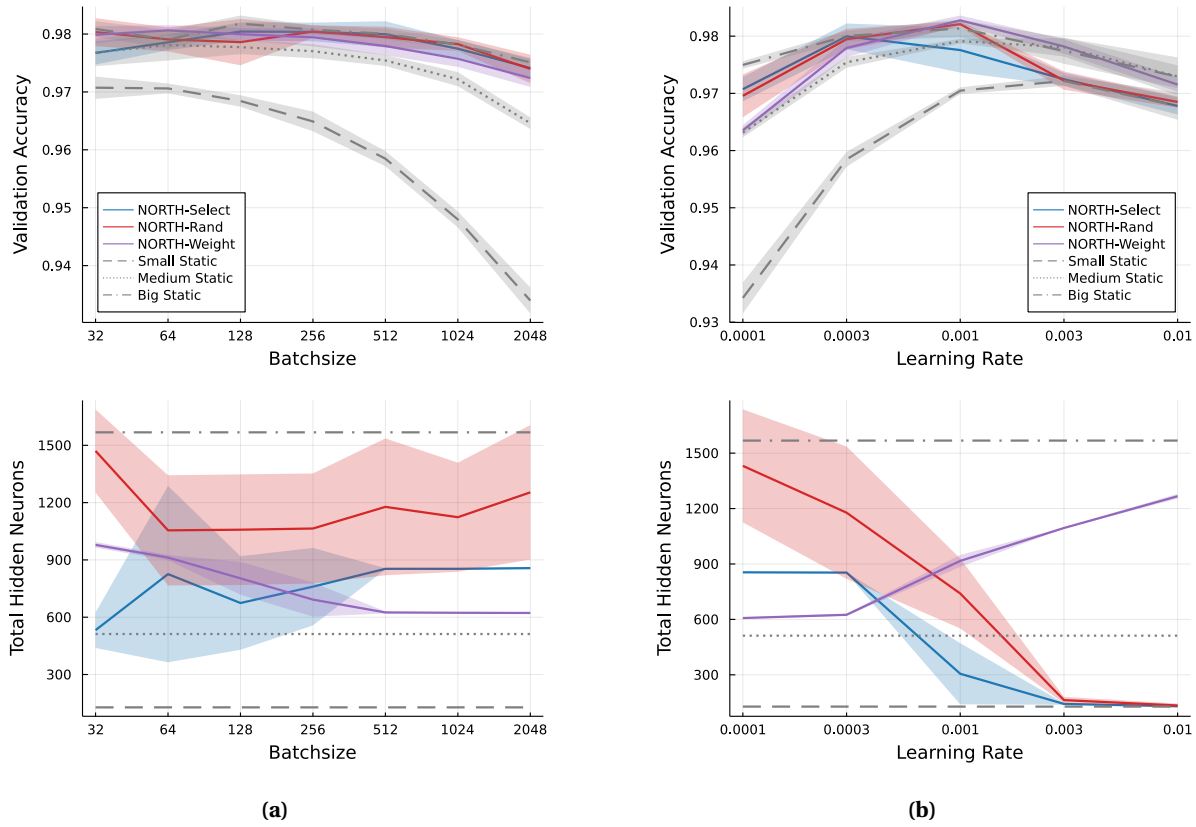


(a) Accuracy versus network size.



(b) Training time versus network size.

Figure 33: Comparison of validation performance and sensitivity to γ_a of ϕ_a^{ED} and ϕ_a^{OG} in activation-based strategies.



(a) **(b)**

Figure 34: Effects of (a) batchsize and (b) learning rate on NORTH-Select and NORTH-Random, compared to dynamic and static baselines.

7 Growing and Pruning ANNs

The preceding work on filling foundational gaps in structural learning has naturally led to investigation beyond the simple paradigm of statically distributed supervised learning of a single task from scratch, which provides an *in vitro* test bed for developing new techniques, towards more comprehensive and realistic problems. The typical paradigm of hand-designing staic architectures is not scalable with the growing desire to apply ANNs in more complex and even dynamic tasks, where the training data distribution may change over time. The simplest form of a dynamic learning environment is transfer learning, where an ANN may be generally pre-trained on a source dataset before fine-tuning a specific copy on the target dataset, improving computational reuse and even performance on the target task. Fine-tuning usually consists of the same generic training approach of gradient descent with a static architecture.

As intermediate activations are effective for out-of-distribution (OoD) detection [268], the inverse suggests they could also inform neural growth [20] and pruning. During fine-tuning on the target task, some existing features may be more useful or adaptable to the target task than others: neurons that are redundant with others or show OoD activation patterns such as dormancy [269] may be pruned and new neurons with incoming distribution-aware initialization may be added. This strategically adapts the model to the new domain by augmenting the most useful components of the inductive bias transferred from pre-training.

We propose and investigate the utility of growing and pruning neurons during transfer learning, resulting in a fine-tuned model with a customized architecture for the target task. We study the use of informed scheduling for the growth and pruning operations, removing the hand-designed schedules and algorithmic details seen in other comparable approaches. We define the DYNO (**Dy**namic **N**eural **O**ptimization) grow-and-prune algorithm: the proposed strategies that comprise DYNO only use activation information from just the forward pass, without any masking or gradient calculations. We study DYNO in multiple transfer learning scenarios, analyzing how different layers respond in across various transfer contexts. DYNO yields dynamically shaped efficient models with tuned layer widths and tuned parameters, without predetermined architectural scheduling.

7.1 Problem Definition

We adopt the same notation as Section 6.1. An artificial neural network (ANN) f may be optimized through empirical risk minimization of a loss function L for a dataset D consisting of inputs x and outputs y:

$$\arg\min_{f} \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim D}\, L(f(\boldsymbol{x}),\boldsymbol{y}). \tag{36}$$

For a dense multi-layer perceptron (MLP) with d hidden layers, f may be expressed as $f(\boldsymbol{x}) = \sigma_{d+1}(\boldsymbol{W}_{d+1}\sigma_d(...\boldsymbol{W}_2\sigma_1(\boldsymbol{W}_1\boldsymbol{x})...))$, where σ_l is a nonlinear activation function that also adds a row for the bias, and $\boldsymbol{W}_l \in \mathbb{R}^{M_l\times(M_{l-1}+1)}$ is the weight matrix for the l^{th} layer, including the bias parameters. We define $\boldsymbol{h}_l = \sigma_l(\boldsymbol{W}_l\sigma_{l-1}(...\boldsymbol{W}_2\sigma_1(\boldsymbol{W}_1\boldsymbol{x})...))$ as the post-activations of layer l. For n samples, $\boldsymbol{H}_l \in \mathbb{R}^{M_l\times n}$ is the post-activation matrix.

For convolutional layers, we consider that a channel is analogous to a neuron in a dense layer. However, both the activation for a channel and a single sample as well as the parameterization of the connection between two channels are matrices instead of single values. Thus, $\boldsymbol{H}_l \in \mathbb{R}^{H_l\times W_l\times M_l\times n}$ and $\boldsymbol{W}_l \in \mathbb{R}^{k_l\times k_l\times M_l\times(M_{l-1}+1)}$.

Standard ANN training procedures pre-define the size and structure of the weight matrices W, updating their parameters via stochastic gradient descent over mini-batches sampled from the current dataset towards optimizing Equation (36).

Algorithm 5 A general framework for neural grow-and-prune optimization.

procedure GROW-AND-PRUNE($\text{TRIG}_{\text{GROW}}$, $\text{INIT}_{\text{GROW}}$, $\text{TRIG}_{\text{PRUNE}}$, $\text{SELECT}_{\text{PRUNE}}$, initial ANN f)
  while f not converged **do**
    for each hidden layer l **do**
      if $\text{TRIG}_{\text{PRUNE}}(f,l) > 0$ **then**
        Remove $\text{TRIG}_{\text{PRUNE}}(f,l)$ neurons by $\text{SELECT}_{\text{PRUNE}}(f,l)$
      if $\text{TRIG}_{\text{GROW}}(f,l) > 0$ **then**
        Add $\text{TRIG}_{\text{GROW}}(f,l)$ neurons using $\text{INIT}_{\text{GROW}}(f,l)$
    for n steps **do**
      Gradient descent step on current existing weights and dataset
  return trained f

To perform neurogenesis or neural pruning, the addition or removal of k neurons to the l^{th} layer is accomplished by appending or removing k rows of fan-in weights in $\boldsymbol{W}_l$ and k columns of fan-out weights in $\boldsymbol{W}_{l+1}$. Utilizing these operations in the empirical risk minimization of Equation (36) defines the neural grow-and-prune optimization. We restrict the search space of where to add new neurons to within existing layers.

7.2 Grow and Prune for Transfer Learning

Our general proposed framework for neural grow-and-prune is presented in Algorithm 5. This framework cycles through each optimization operation of pruning, growth, and parameter optimization via gradient descent, allowing the grow-and-prune details such as the trigger strategies, growth initialization strategy, and pruning selection strategies to be defined alongside other training details such as the gradient descent optimizer. We propose the **Dy**namic **N**eural **O**ptimization (DYNO) grow-and-prune algorithm with the strategies defined as follows.

Following [20], we define triggers and initialization/selection strategies for each operation of growing neurons and pruning neurons. The trigger is evaluated regularly to determine, for each layer, how many neurons to grow or prune. The initialization strategy for growing or selection strategy is then applied to carry out the structural change.

The trigger for both operations is based on the orthogonality metric of *effective dimensionality*, the ϵ-numerical rank of the post-activation matrix [253, 257]. For layer l and n samples generating the post-activation $\boldsymbol{H}_l$, the effective dimension metric ϕ_{ED} may be estimated by

$$\phi_{ED}(f,l) = \frac{1}{M_l}\left|\left\{\sigma \in \text{SVD}\left(\frac{1}{\sqrt{n}}\boldsymbol{H}_l\right) \middle| \sigma > \epsilon\right\}\right|, \tag{37}$$

where $\text{SVD}\left(\frac{1}{\sqrt{n}}\boldsymbol{H}_l\right)$ is the set of singular values of $\frac{1}{\sqrt{n}}\boldsymbol{H}_l$ and $\epsilon > 0$ is a small threshold. For convolutional layers, $\boldsymbol{H}_l$ is flattened to $\mathbb{R}^{M_l \times (H_l W_l n)}$, so that the metric is relative to the number of neurons or channels in layer l.

A significant increase in the orthogonality metric is used to trigger neurogenesis, while a significant decrease is used to trigger pruning:

$$\text{TRIG}_{\text{PRUNE}}(f,l) = \min\left(0, \left\lfloor M_l\left((1-\delta)\phi^*_{ED} - \phi_{ED}(f,l)\right)\right\rfloor\right), \tag{38}$$

$$\text{TRIG}_{\text{GROW}}(f,l) = \min\left(0, \left\lfloor M_l\left(\phi_{ED}(f,l) - (1+\delta)\phi^*_{ED}\right)\right\rfloor\right), \tag{39}$$

where $\delta > 0$ is a small threshold and ϕ^*_{ED} is the value of $\phi_{ED}(f,l)$ at the last structural change of f. This combination of trigger strategies slims the layer when its activations are expressible in fewer

dimensions, and expands it when the feature space in near expressible capacity. The updating baseline allows structural operations to occur dynamically in response to changes in the training distribution, with minimal dependence on the frequency of trigger evaluation.

Neurons to prune are selected based on a greedy cosine similarity algorithm, $\text{SELECT}_{\text{PRUNE}}(f, l)$. In the case of ReLU-activated layers, all completely dormant neurons are first pruned. Then, until the total number of neurons to prune is reached, the neuron with the highest norm of cosine similarities of normalized post-activations with other remaining neurons is greedily pruned.

New neurons are grown with the initialization strategy, $\text{INIT}_{\text{GROW}}(f, l)$, as in NORTH-Select [20]: from candidates with scaled random fan-in vectors and zeroed fan-out vectors, select neurons that independently maximize $\phi_{ED}(f, l)$ with the existing neurons in that layer.

7.3 Experiments

We apply neural grow-and-prune transfer learning on three classification scenarios, each beginning with VGG11 pretrained on Imagenet [270]. The target datasets are:

- **Imagenette:** 10 easily distinguishable classes of Imagenet [271], thus making high-level features useful but not detailed features that may distinguish more similar classes.
- **Imagewoof:** 10 dog breed classes of Imagenet [271], thus requiring minute details within a subspace of the original distribution.
- **Galaxy10:** satellite images of galaxies in 10 shape-based categories [216], so the underlying distribution is very different from natural images as in Imagenet.

We additionally implemented Surgical Fine-Tuning (SFT) [272], which dynamically adjusts layer-wise learning rates based on the layer's relative gradient norm, as an orthogonal approach to transfer learning.

All experiments begin with the same Imagenet-pretrained VGG11 backbone, with standard layer widths. The results show the mean, standard deviation, and (when applicable) covariance across 5 trials each with different random seeds. For Galaxy10, the final classification layer is replaced with a reinitialized 10-way classification layer. For Imagenette and Imagewoof, the final classification layer is replaced with a 10-way classification layer consisting of the 10 respective classification neurons corresponding to each class. All trials are run for 1000 iterations with 64 samples per iteration, performing pruning and growth trigger evaluations every 50 iterations. The effective dimensionality threshold ϵ is 1e-2 and the trigger threshold δ is 3e-2. The optimizer is Adam with a learning rate of 3e-4 and weight decay of 1e-3. All metrics are evaluated on post-activations, which are tracked for each layer in a FIFO buffer that is dynamically sized to contain more samples than neurons, as necessary for the effective dimensionality calculation. Only forward pass information is required. The number of candidates generated for neural growth is $100 \times \text{TRIG}_{\text{GROW}}(f, l)$. ImageNet, Imagenette, and Imagewoof are available freely for research purposes. Galaxy10 is available under the MIT license.

The results of applying DYNO compared to standard static fine-tuning are shown in Figure 35. When comparing test accuracy to inference FLOPs, the fine-tuned models have slightly lower average accuracy but are consistently more efficient in compute cost. Network sizes are consistent within tasks: models for Galaxy10 are about half as expensive as the original pre-trained model. DYNO causes immediately large architectural changes, then progressively smaller later in training, approaching architectural convergence but allowing small amounts of turnover well after the distribution change. Considering layer-wise architectural changes of DYNO, models across all three tasks had most change

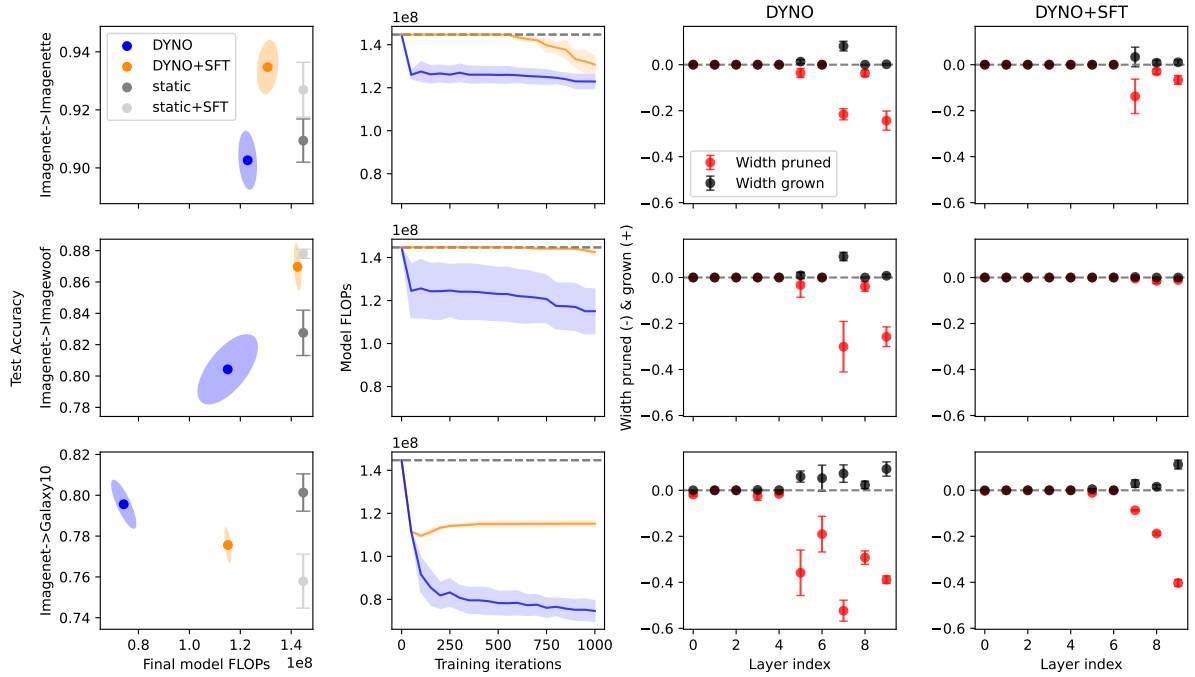


Figure 35: Experimental results for transfer to Imagenette (top), Imagewoof (middle), and Galaxy10 (bottom), including combinations of DYNO and SFT [272]. First column: final test accuracy against final model inference FLOPs. Second column: progression of inference cost over the course of training. Third and fourth columns: total neurons grown and pruned per layer over the course of DYNO (third column) and DYNO+SFT (fourth column), relative to the original layer width. All covariance ellipses, error clouds, and error bars show standard deviation across the 5 trials.

occur in the later layers of the network, particularly the final convolution (layer 7) and final dense layer before the classification layer (layer 9). Models for Galaxy10 had significantly more architectural change, both for pruning and growing, and in more layers.

To fine-tune on a similar task, such as transferring to Imagenette, DYNO made very few changes, similar to what would happen with standard pruning of a final layers. However, when distribution shifts, such as transferring to Galaxy10, the architecture changes throughout: major modifications are made in the final layer, and there are even structural modifications in the early layers. This shows that DYNO responds to distribution shift based on the learned task to appropriately modify the architecture.

DYNO was not able to surpass the test accuracy of larger static models. This may be due to the simple linear schedule of architectural evaluation without a long final tuning phase or any learning rate decay, nor extensive hyperparameter tuning. Such "tricks" are often employed for improving performance but may confound results so thus were avoided in this preliminary study.

The strategies of DYNO are intended to minimize the number and impact of hyperparameters. For example, the moving baseline used in the trigger functions reduces dependence on the trigger evaluation frequency. Further work on hyperparameter optimization and specialization is warranted, such as learnable thresholds [273] and layer-wise thresholds, which could improve test accuracy compared to static fine-tuning.

Towards structurally heterogeneous learning rates, we additionally reimplemented Surgical Fine-Tuning (SFT) [272] as an orthogonal transfer learning tool. We specifically use their Auto-RGN method, which dynamically scales the learning rate of each layer based on the relative gradient norm. The experimental results are shown in Figure 35. All hyperparameters were kept the same, with SFT hyperparameters used from Lee et al. [272].

With SFT, DYNO performs as well or better than static architectures across all tasks. However, DYNO+SFT has significantly less architectural change: virtually none during transfer to Imagewoof, only late changes for Imagenette, and immediate pruning but then subtle growth for Galaxy10. A potential explanation is that SFT reduces some signaled need for grow-and-prune by heterogeneously and dynamically tweaking learning rates. Further hyperparameter tuning towards synergy could yield improved results: implementing SFT on a neuronal level could be even more beneficial.

A notable potential confounder is that SFT only reduces the learning rate from the baseline value used, so all trials using SFT had smaller learning rates overall. This explains why performance was improved for Imagenette and Imagewoof, which are very close in distribution to Imagenet, but was worse for Galaxy10.

7.4 Discussion

The DYNO grow-and-prune algorithm shows dynamic layer-specific responsiveness for architectural changes during fine-tuning without the need for any hand-engineered bias, such has predetermined schedules of architectural change. This is a benefit of dynamically informed scheduling and minimal hyper-parameterization. This direction is important for the motivation of AutoML towards more automation and less human engineering.

Grow-and-prune algorithms have a very complex design space due to the interplay of growing, pruning, and training a model. This preliminary study only considered one intuition-guided instance within this space by defining each of $\text{TRIG}_{\text{GROW}}$, $\text{INIT}_{\text{GROW}}$, $\text{TRIG}_{\text{PRUNE}}$, and $\text{SELECT}_{\text{PRUNE}}$, but the vastness prompts algorithmic optimization to find useful combinations of strategies. Previous automated algorithmic discovery methods such as [274, 275] could be extended to include neural growth and pruning as potential operators during training.

Transfer learning is the first step from static supervised learning: further in the same direction include continual learning and multi-task learning. These dynamic paradigms benefit from domain generalization and avoiding catastrophic forgetting, which can be supported architecturally. Architectural optimization could help find efficient parameter sharing between distributions or tasks. Grow-and-prune may naturally be effective for alleviating the decay in plasticity normally observed in ANNs [276]. To avoid catastrophic forgetting, the pruning mechanism may be modified: activation information could be used to mark which existing neurons could either be used with locked parameters or be copied and fine-tuned. These architectural techniques may additionally detect new contexts, as demonstrated by [268], to trigger appropriate architectural adjustment, potentially incorporated with structurally aware representation disentanglement. These in tandem could ameliorate problems such as representation collapse [253] and lack of generalization [277], in addition to finding dynamically constructed architectures for improved efficiency.

7.5 Additional Related Works

Previous neural grow-and-prune works often focus on the "where" and "how" of pruning, using generic schedules and neurogenesis initializations. The most comparable works that separately consider informed neural creation and pruning are Firefly [110] and NeST [68], which use gradient-based information. Further works perform less informed neural creation [101, 278] or ephemeral pruning via masking [117]. [279] studies synaptic grow-and-prune for incremental learning and [280] performs unstructured pruning during fine-tuning, or fine-pruning. Structured fine-pruning has mostly been studied in the case of large language models (LLMs) driven by their prohibitively large training and inference costs when unpruned [281]. [282] performs structured pruning driven by redundancy metrics. No other known neural grow-and-prune works that allow dynamic layer widths consider the transfer learning case nor use only activation-based information.

7.6 Conclusion

Grow-and-prune during transfer learning and other dynamic learning environments is an intuitively useful yet challenging addition that warrants further investigation. With DYNO, we propose a framework for triggering pruning and growing strategies and demonstrate that this can result in structural changes during the learning process which respond to a downstream task distribution. This work synthesizes dynamic scheduling from Sections 5 and 6 and neural creation from Section 6 towards dynamic structural adjustment during distribution shift, beginning with structural adaptation for the simple case of transfer learning.

8 Discussion

My experimental work has thus far focused on improving aspects of structural learning that are not well studied as identified via literature analysis for structural learning (Section 3), particularly extending NAS to architectural representations of equivariance (Section 4), scheduling and improving differentiable NAS (Section 5), isolating and improving methods for neural creation (Section 6), and studying the interplay of neural growth and pruning in dynamic learning environments (Section 7).

These contributions provide tools for the automation of architectural optimization at various structural levels, particularly focusing on more informed scheduling and expanded algorithmic applications. Scheduling, which comprises what structural changes to perform and when, where, how, and why to perform them, is important for achieving automation and adaptivity. Contributions in this manuscript, particularly of Sections 5-7, utilize dynamic heuristics of learning to inform scheduling so that these questions may be automatically answered. Expanding the applicability of structural learning is crucial for escaping the tight bounds of well-studied yet artificial benchmarks towards more realistic problems. Section 4 directly accomplishes this through a novel domain of structures that yield optimizably partial equivariance, while Section 6 investigates novel methods for adding to architectures, rather than just pruning from them. As Section 7 begins to demonstrate, the proposed tools fill multiple foundational gaps in structural learning such that methods may now be synergistically combined towards comprehensive and adaptive systems with real-world applications.

The more immediate and practical goals of this research direction are towards automated machine learning. The preceding chapters focus on dynamic architectural design, which can work synergistically with automated hyperparameter tuning and even algorithmic optimization. This path may eventually go towards automation of machine learning as a tool: rather than an expert machine learning engineer hand-designing the architecture, hyperparameters, and other aspects of the pipeline, a scientist could directly get a trained structurally-optimized model simply by defining their problem and providing potential data in an intuitive programming language.

Other long-term goals of this work are to break the convenient yet restricting conventions of deep learning towards more organic methods. The presented contributions are loosely inspired by neuroscience: the human brain achieves the best-known example of human-defined intelligence. However, the brain is still a dark gray box to us: we've only scratched the surface in elucidating its learning mechanisms even with exponential progress in neural imaging and modeling. The fields of neuroscience and neuro-inspired artificial intelligence may work interactively towards furthering the both understanding and capabilities of intelligent systems, whether biological or artificial.

Sparsity is a multi-dimension spectrum that overlaps with structure and priors: it can occur at different levels and even implicitly, such as the relative sparsity of convolutions or the dynamic sparsity of attention. The organization, modularity, and other human-comprehensible properties of such ANN structures allows us to work with more orders of magnitude of dimensions at a time. This has already enabled models up to billions of parameters in size but in a human-constrained search space, which contains only known structures and feasibly simple patterns of connectivity. The beauty of architectural optimization is potentially breaking these patterns beyond what humans can conceive, although at the possible, yet temporary, cost of interpretability.

8.1 Future Work

The cumulation of these contributions within the context of the current state of literature provides a basis for the upcoming interdisciplinary research directions presented here, in addition to those specifically discussed within each chapter.

8.1.1 Structural Plasticity in Reinforcement Learning

Reinforcement learning (RL) removes the assumption of immediate supervision, in the context of an ANN-controlled agent acting over time within an environment. Distribution shift may occur when the agent encounters a new domain of states, such as a new level of a video game or a faulty sensor. Applying a structural grow-and-prune algorithm, such as extending the work of Section 7, to this context could allow the agent to be more flexible and adapt to the given environment, balancing plasticity with stability. However, the RL paradigm poses additional challenges of measuring heuristics and gradient information with less supervision than in fully-supervised tasks.

8.1.2 Dynamic Equivariance Search

The tools and algorithms presented in Section 4 permit the relaxation of equivariance constraints during architectural optimization. By further developing a complementary operation to increase equivariance constraints, the discrete structural equivariance level could be searched more dynamically, in a similar fashion as neural pruning complementing neural creation in Section 7. Such an operation may utilize an equivariance distance metric both for scheduling as well as to determine the optimal projection to equivariance of a supergroup. It would necessarily not be a function-preserving morphism, similar to neural pruning. This equivariance-constraining operation could be used in tandem with the equivariance relaxation morphism in an evolutionary NAS algorithm similar to EquiNAS$_E$, or in an algorithm more akin to that presented in Section 7 using scheduling heuristics such as an equivariance distance metric for each operation.

8.1.3 Structural Learning for Transformers

Most existing architectural optimization works have focused on architectural search spaces for relatively simple computer vision tasks. However, machine learning in the 2020s is shifting towards models adept at complex or even multi-modal structures of data. The attention-based transformer architecture has become the state-of-the-art for many domains, from simple computer vision to much more complex tasks. Structurally augmenting such models requires defining appropriate search spaces and adjusting the algorithms for navigating them efficiently due to their massive size. The grow-and-prune work of Section 7 may be extended to investigate transformers, where pre-training is standard practice: the structural strategies can be directly applied to different components such as hidden neurons, attention heads and matrix columns, and even entire blocks. This, along with extending geometric techniques as mentioned in Section 8.1.2 will further empower architectural search spaces on current hot topics such as specializing large language models (LLMs) and building geometric models for complex structured data and tasks.

8.1.4 Woke LLMs: Equivariance Towards Socially Unbiased Models

At the intersection of the preceding directions is evading social bias in LLM training [283]: the contributions towards optimizing partial equivariance from Section 4 may be extended towards achieving theoretically valid appropriate symmetry of LLMs to representations of social issues, being aware of constructs such as gender, race, and sexual orientation without being blind nor biased to them. The architectural constraints may be algorithmically proposed and interactively guided by human feedback, rather than completely learned from training data, to ensure latent biases in data are not learned.

8.2 Conclusion

Structural learning is a unique sub-field of machine learning with close ties to neuroscience, posited towards further automation and increasing artificial intelligence through the concurrent optimization of ANN architectures with their parameters. The work within this manuscript progresses our

understanding of structural learning, particularly dynamic scheduling and expanded tools to effectuate structural change across various granularities of the architecture of an ANN, towards improved automation, adaptivity, and applicability. These contributions complete a foundation for structural learning, empowering the automation of machine learning towards more powerful and adaptable models.

References

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[2] G. E. Moore *et al.*, "Progress in digital integrated electronics," in *Electron devices meeting*, vol. 21. Washington, DC, 1975, pp. 11–13.

[3] H. N. Mhaskar and T. Poggio, "Deep vs. shallow networks: An approximation theory perspective," *Analysis and Applications*, vol. 14, no. 06, pp. 829–848, 2016.

[4] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao, "Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review," *International Journal of Automation and Computing*, vol. 14, no. 5, pp. 503–519, 2017.

[5] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[6] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[7] B. G. Rash, J. B. Ackman, and P. Rakic, "Bidirectional radial Ca2+ activity regulates neurogenesis and migration during early cortical column formation," *Science advances*, vol. 2, no. 2, p. e1501733, 2016.

[8] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] T. S. Cohen, M. Geiger, and M. Weiler, "A general theory of equivariant cnns on homogeneous spaces," *Advances in neural information processing systems*, vol. 32, 2019.

[11] L. J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Proceedings of the 27th International Conference on Neural Information Processing Systems-Volume 2*, 2014, pp. 2654–2662.

[12] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *International Conference on Learning Representations*, 2018.

[13] D. Zou, Y. Cao, D. Zhou, and Q. Gu, "Gradient descent optimizes over-parameterized deep relu networks," *Machine Learning*, vol. 109, no. 3, pp. 467–492, 2020.

[14] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, p. 106, 1962.

[15] S. Kanjlia, C. Lane, L. Feigenson, and M. Bedny, "Absence of visual experience modifies the neural basis of numerical thinking," *Proceedings of the National Academy of Sciences*, vol. 113, no. 40, pp. 11 172–11 177, 2016.

[16] K. Maile, H. Luga, and D. G. Wilson, "Structural learning in artificial neural networks: A neural operator perspective," *Transactions on Machine Learning Research*, 2022.

[17] K. Maile, D. G. Wilson, and P. Forré, "Equivariance-aware architectural optimization of neural networks," *11th International Conference on Learning Representations*, 2023.

[18] K. Maile, E. Lecarpentier, H. Luga, and D. G. Wilson, "DARTS-PRIME: Regularization and scheduling improve constrained optimization in differentiable NAS," *arXiv preprint arXiv:2106.11655*, 2021.

[19] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[20] K. Maile, E. Rachelson, H. Luga, and D. G. Wilson, "When, where, and how to add new neurons to anns," in *International Conference on Automated Machine Learning*. PMLR, 2022, pp. 18–1.

[21] K. Maile, H. Luga, and D. G. Wilson, "Neural growth and pruning in dynamic learning environments," *International Conference on Automated Machine Learning*, 2023.

[22] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[24] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.

[25] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *arXiv preprint arXiv:1702.05659*, 2017.

[26] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[27] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[28] Y. Bengio, D.-H. Lee, J. Bornschein, T. Mesnard, and Z. Lin, "Towards biologically plausible deep learning," *arXiv preprint arXiv:1502.04156*, 2015.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[30] T. M. Hospedales, A. Antoniou, P. Micaelli, and A. J. Storkey, "Meta-learning in neural networks: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[31] X. He, K. Zhao, and X. Chu, "AutoML: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[32] N. Cook, "The neuron-level phenomena underlying cognition and consciousness: Synaptic activity and the action potential," *Neuroscience*, vol. 153, no. 3, pp. 556–570, 2008.

[33] R. D. Fields, A. Araque, H. Johansen-Berg, S.-S. Lim, G. Lynch, K.-A. Nave, M. Nedergaard, R. Perez, T. Sejnowski, and H. Wake, "Glial biology in learning and cognition," *The neuroscientist*, vol. 20, no. 5, pp. 426–431, 2014.

[34] H. Kennedy, D. C. Van Essen, and Y. Christen, *Micro-, meso-and macro-connectomics of the brain*. Springer Nature, 2016.

[35] A. M. Fjell and K. B. Walhovd, "Structural brain changes in aging: courses, causes and cognitive consequences," *Reviews in the Neurosciences*, vol. 21, no. 3, pp. 187–222, 2010.

[36] G. Radnikow and D. Feldmeyer, "Layer-and cell type-specific modulation of excitatory neuronal activity in the neocortex," *Frontiers in neuroanatomy*, vol. 12, p. 1, 2018.

[37] G. Z. Tau and B. S. Peterson, "Normal development of brain circuits," *Neuropsychopharmacology*, vol. 35, no. 1, pp. 147–168, 2010.

[38] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, 1990, pp. 598–605.

[39] C. E. Martin and P. K. Pilly, "Probabilistic Program Neurogenesis," in *The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE)*. MIT Press, 2019, pp. 440–447.

[40] P. Li, I. Farkas, and B. MacWhinney, "Early lexical development in a self-organizing neural network," *Neural Networks*, vol. 17, no. 8, pp. 1345–1362, 2004, new Developments in Self-Organizing Systems. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608004001534

[41] R. Aljundi, F. Babiloni, M. Elhoseiny, M. Rohrbach, and T. Tuytelaars, "Memory aware synapses: Learning what (not) to forget," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 139–154.

[42] F. A. Azevedo, L. R. Carvalho, L. T. Grinberg, J. M. Farfel, R. E. Ferretti, R. E. Leite, W. J. Filho, R. Lent, and S. Herculano-Houzel, "Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain," *Journal of Comparative Neurology*, vol. 513, no. 5, pp. 532–541, 2009.

[43] S. Malik, G. Vinukonda, L. R. Vose, D. Diamond, B. B. Bhimavarapu, F. Hu, M. T. Zia, R. Hevner, N. Zecevic, and P. Ballabh, "Neurogenesis continues in the third trimester of pregnancy and is suppressed by premature birth," *Journal of neuroscience*, vol. 33, no. 2, pp. 411–423, 2013.

[44] N. Urbán and F. Guillemot, "Neurogenesis in the embryonic and adult brain: same regulators, different roles," *Frontiers in cellular neuroscience*, vol. 8, p. 396, 2014.

[45] W. Wu, K. Wong, J.-h. Chen, Z.-h. Jiang, S. Dupuis, J. Y. Wu, and Y. Rao, "Directional guidance of neuronal migration in the olfactory system by the protein slit," *Nature*, vol. 400, no. 6742, pp. 331–336, 1999.

[46] S. Ackerman, "The development and shaping of the brain," *Discovering the Brain*, pp. 86–103, 1992.

[47] G.-l. Ming and H. Song, "Adult neurogenesis in the mammalian brain: Significant answers and significant questions," *Neuron*, vol. 70, no. 4, pp. 687–702, 2011.

[48] A. Ernst, K. Alkass, S. Bernard, M. Salehpour, S. Perl, J. Tisdale, G. Possnert, H. Druid, and J. Frisén, "Neurogenesis in the striatum of the adult human brain," *Cell*, vol. 156, no. 5, pp. 1072–1083, 2014.

[49] H. T. Ghashghaei, C. Lai, and E. Anton, "Neuronal migration in the adult brain: are we there yet?" *Nature Reviews Neuroscience*, vol. 8, no. 2, pp. 141–151, 2007.

[50] R. Mozzachiodi and J. H. Byrne, "More than synaptic plasticity: role of nonsynaptic plasticity in learning and memory," *Trends in neurosciences*, vol. 33, no. 1, pp. 17–26, 2010.

[51] C. L. Waites, A. M. Craig, and C. C. Garner, "Mechanisms of vertebrate synaptogenesis," *Annu. Rev. Neurosci.*, vol. 28, pp. 251–274, 2005.

[52] L. Erskine and E. Herrera, "The retinal ganglion cell axon's journey: Insights into molecular mechanisms of axon guidance," *Developmental biology*, vol. 308, no. 1, pp. 1–14, 2007.

[53] C. Pfeiffenberger, T. Cutforth, G. Woods, J. Yamada, R. C. Rentería, D. R. Copenhagen, J. G. Flanagan, and D. A. Feldheim, "Ephrin-As and neural activity are required for eye-specific patterning during retinogeniculate mapping," *Nature neuroscience*, vol. 8, no. 8, p. 1022, 2005.

[54] T. J. Nelson and D. L. Alkon, "Molecular regulation of synaptogenesis during associative learning and memory," *Brain research*, vol. 1621, pp. 239–251, 2015.

[55] W. Kelsch, S. Sim, and C. Lois, "Watching synaptogenesis in the adult brain," *Annual review of neuroscience*, vol. 33, pp. 131–149, 2010.

[56] M. M. Riccomagno and A. L. Kolodkin, "Sculpting neural circuits by axon and dendrite pruning," *Annual review of cell and developmental biology*, vol. 31, pp. 779–805, 2015.

[57] D. O. Hebb, *The organisation of behaviour: a neuropsychological theory*. Science Editions New York, 1949.

[58] L. C. Katz and C. J. Shatz, "Synaptic activity and the construction of cortical circuits," *Science*, vol. 274, no. 5290, pp. 1133–1138, 1996.

[59] D. H. Hubel, T. N. Wiesel, S. LeVay, H. B. Barlow, and R. M. Gaze, "Plasticity of ocular dominance columns in monkey striate cortex," *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, vol. 278, no. 961, pp. 377–409, 1977.

[60] D. K. Wilton, L. Dissing-Olesen, and B. Stevens, "Neuron-glia signaling in synapse elimination," *Annual review of neuroscience*, vol. 42, pp. 107–127, 2019.

[61] B. Stevens, N. J. Allen, L. E. Vazquez, G. R. Howell, K. S. Christopherson, N. Nouri, K. D. Micheva, A. K. Mehalow, A. D. Huberman, B. Stafford *et al.*, "The classical complement cascade mediates cns synapse elimination," *Cell*, vol. 131, no. 6, pp. 1164–1178, 2007.

[62] D. L. Bishop, T. Misgeld, M. K. Walsh, W.-B. Gan, and J. W. Lichtman, "Axon branch removal at developing synapses by axosome shedding," *Neuron*, vol. 44, no. 4, pp. 651–661, 2004.

[63] A. W. Harrington and D. D. Ginty, "Long-distance retrograde neurotrophic factor signalling in neurons," *Nature Reviews Neuroscience*, vol. 14, no. 3, pp. 177–187, 2013.

[64] P. Vanderhaeghen and H.-J. Cheng, "Guidance molecules in axon pruning and cell death," *Cold Spring Harbor perspectives in biology*, vol. 2, no. 6, p. a001859, 2010.

[65] W. Yeo and J. Gautier, "Early neural cell death: dying to become neurons," *Developmental biology*, vol. 274, no. 2, pp. 233–244, 2004.

[66] R. R. Buss, W. Sun, and R. W. Oppenheim, "Adaptive roles of programmed cell death during nervous system development," *Annu. Rev. Neurosci.*, vol. 29, pp. 1–35, 2006.

[67] M. Fricker, A. M. Tolkovsky, V. Borutaite, M. Coleman, and G. C. Brown, "Neuronal cell death," *Physiological reviews*, vol. 98, no. 2, pp. 813–880, 2018.

[68] X. Dai, H. Yin, and N. K. Jha, "NeST: A neural network synthesis tool based on a grow-and-prune paradigm," *IEEE Transactions on Computers*, vol. 68, no. 10, pp. 1487–1497, 2019.

[69] M. Kang and B. Han, "Operation-aware soft channel pruning using differentiable masks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5122–5131.

[70] P. T. Quinlan, "Structural change and development in real and artificial neural networks," *Neural Networks*, vol. 11, no. 4, pp. 577–599, 1998.

[71] T. Elsken, J.-H. Metzen, and F. Hutter, "Simple and efficient architecture search for convolutional neural networks," *arXiv preprint arXiv:1711.04528*, 2017.

[72] Y. Ci, C. Lin, M. Sun, B. Chen, H. Zhang, and W. Ouyang, "Evolving search space for neural architecture search," *arXiv preprint arXiv:2011.10904*, 2020.

[73] Z. Lu, K. Deb, E. Goodman, W. Banzhaf, and V. N. Boddeti, "Nsganetv2: Evolutionary multi-objective surrogate-assisted neural architecture search," in *European Conference on Computer Vision*. Springer, 2020, pp. 35–51.

[74] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies." *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[75] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[76] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[77] G. W. Lindsay, "Attention in psychology, neuroscience, and machine learning," *Frontiers in computational neuroscience*, p. 29, 2020.

[78] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[79] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," *arXiv preprint arXiv:2102.00554*, 2021.

[80] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *Journal of Machine Learning Research*, vol. 20, pp. 1–21, 2019.

[81] L. Xie, X. Chen, K. Bi, L. Wei, Y. Xu, Z. Chen, L. Wang, A. Xiao, J. Chang, X. Zhang *et al.*, "Weight-sharing neural architecture search: A battle to shrink the optimization gap," *arXiv preprint arXiv:2008.01475*, 2020.

[82] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 24–35, 2019.

[83] J. F. Miller, "Improbed: Multiple problem-solving brain via evolved developmental programs," *Artificial Life*, vol. 27, no. 3–4, pp. 300–335, 2022.

[84] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural Networks*, 2019.

[85] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, and Y. Wang, "Dynamic neural networks: A survey," *arXiv preprint arXiv:2102.04906*, 2021.

[86] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[87] B. R. Bartoldson, A. S. Morcos, A. Barbu, and G. Erlebacher, "The generalization-stability tradeoff in neural network pruning," *arXiv preprint arXiv:1906.03728*, 2019.

[88] L. Liebenwein, C. Baykal, B. Carter, D. Gifford, and D. Rus, "Lost in pruning: The effects of pruning neural networks beyond test accuracy," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 93–138, 2021.

[89] E. Dayan and L. G. Cohen, "Neuroplasticity subserving motor skill learning," *Neuron*, vol. 72, no. 3, pp. 443–454, 2011.

[90] S. Liu, D. C. Mocanu, A. R. R. Matavalam, Y. Pei, and M. Pechenizkiy, "Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware," *Neural Computing and Applications*, vol. 33, pp. 2589–2604, 2021.

[91] J. Peng, B. Tang, H. Jiang, Z. Li, Y. Lei, T. Lin, and H. Li, "Overcoming long-term catastrophic forgetting through adversarial neural pruning and synaptic consolidation," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[92] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature communications*, vol. 9, no. 1, p. 2383, 2018.

[93] G. Bellec, D. Kappel, W. Maass, and R. Legenstein, "Deep rewiring: Training very sparse deep networks," *International Conference on Learning Representations*, 2018.

[94] M. Wortsman, A. Farhadi, and M. Rastegari, "Discovering neural wirings," *Advances in Neural Information Processing Systems*, vol. 32, pp. 2684–2694, 2019.

[95] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *IEEE international conference on neural networks*. IEEE, 1993, pp. 293–299.

[96] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[97] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," *arXiv preprint arXiv:1608.04493*, 2016.

[98] N. Lee, T. Ajanthan, and P. H. S. Torr, "SNIP: Single-shot Network Pruning based on Connection Sensitivity," *arXiv:1810.02340 [cs]*, feb 2019.

[99] W. J. Puma-Villanueva, E. P. dos Santos, and F. J. Von Zuben, "A constructive algorithm to synthesize arbitrarily connected feedforward neural networks," *Neurocomputing*, vol. 75, no. 1, pp. 14–32, jan 2012.

[100] X. Dai, H. Yin, and N. K. Jha, "Incremental Learning Using a Grow-and-Prune Paradigm with Efficient Neural Networks," *IEEE Transactions on Emerging Topics in Computing*, may 2019. [Online]. Available: http://arxiv.org/abs/1905.10952

[101] X. Du, Z. Li, Y. Ma, and Y. Cao, "Efficient network construction through structural plasticity," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 453–464, 2019.

[102] D. B. Chklovskii, B. Mel, and K. Svoboda, "Cortical rewiring and information storage," *Nature*, vol. 431, no. 7010, pp. 782–788, 2004.

[103] C. Scholl, M. E. Rule, and M. H. Hennig, "The information theory of developmental pruning: Optimizing global network architectures using local synaptic rules," *PLoS computational biology*, vol. 17, no. 10, p. e1009458, 2021.

[104] C. Laurent, C. Ballas, T. George, N. Ballas, and P. Vincent, "Revisiting loss modelling for unstructured pruning," *arXiv preprint arXiv:2006.12279*, 2020.

[105] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *International Conference on Learning Representations*, 2017.

[106] S. Yan, B. Fang, F. Zhang, Y. Zheng, X. Zeng, M. Zhang, and H. Xu, "HM-NAS: Efficient neural architecture search via hierarchical masking," in *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019.

[107] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *International Conference on Learning Representations*, 2017.

[108] A. G. Rust, R. Adams, S. George, and H. Bolouri, "Activity-based pruning in developmental artificial neural networks," in *Proc. of the European Conf. on Artificial Life (ECAL'97)*, 1997, pp. 224–233.

[109] C. Siegel, J. Daily, and A. Vishnu, "Adaptive neuron apoptosis for accelerating deep learning on large scale systems," in *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2016, pp. 753–762.

[110] L. Wu, B. Liu, P. Stone, and Q. Liu, "Firefly neural architecture descent: a general approach for growing neural networks," in *Advances in Neural Information Processing Systems*, 2020.

[111] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 32, 2017.

[112] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[113] P. L. Narasimha, W. H. Delashmit, M. T. Manry, J. Li, and F. Maldonado, "An integrated growing-pruning method for feedforward network training," *Neurocomputing*, vol. 13, no. 71, pp. 2831–2847, 2008.

[114] M. Ye, C. Gong, L. Nie, D. Zhou, A. Klivans, and Q. Liu, "Good subnetworks provably exist: Pruning via greedy forward selection," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 10 820–10 830. [Online]. Available: http://proceedings.mlr.press/v119/ye20b.html

[115] W. Pan, H. Dong, and Y. Guo, "DropNeuron: Simplifying the structure of deep neural networks," *arXiv:1606.07326 [cs, stat]*, jul 2016.

[116] S. Chen, L. Lin, Z. Zhang, and M. Gen, "Evolutionary NetArchitecture Search for deep neural networks pruning," in *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence*, 2019, pp. 189–196.

[117] A. Wan, X. Dai, P. Zhang, Z. He, Y. Tian, S. Xie, B. Wu, M. Yu, T. Xu, K. Chen *et al.*, "Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 965–12 974.

[118] S. Guo, Y. Wang, K. Yuan, and Q. Li, "Differentiable network adaption with elastic search space," *arXiv preprint arXiv:2103.16350*, 2021.

[119] K. A. Laube and A. Zell, "Prune and replace NAS," in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, 2019, pp. 915–921.

[120] X. Li, Y. Zhou, T. Wu, R. Socher, and C. Xiong, "Learn to grow: A continual structure learning framework for overcoming catastrophic forgetting," in *International Conference on Machine Learning*. PMLR, 2019, pp. 3925–3934.

[121] J. Mei, Y. Li, X. Lian, X. Jin, L. Yang, A. Yuille, and J. Yang, "Atomnas: Fine-grained end-to-end neural architecture search," *arXiv preprint arXiv:1912.09640*, 2019.

[122] K. Bi, L. Xie, X. Chen, L. Wei, and Q. Tian, "GOLD-NAS: Gradual, One-Level, Differentiable," *arXiv preprint arXiv:2007.03331*, 2020.

[123] A. Noy, N. Nayman, T. Ridnik, N. Zamir, S. Doveh, I. Friedman, R. Giryes, and L. Zelnik, "Asap: Architecture search, anneal and prune," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020, pp. 493–503.

[124] R. Wang, M. Cheng, X. Chen, X. Tang, and C.-J. Hsieh, "Rethinking architecture selection in differentiable NAS," in *International Conference on Learning Representations*, 2021.

[125] N. Roberts, M. Khodak, T. Dao, L. Li, C. Ré, and A. Talwalkar, "Rethinking neural operations for diverse tasks," *arXiv preprint arXiv:2103.15798*, 2021.

[126] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," *arXiv*, pp. 1–13, dec 2018. [Online]. Available: http://arxiv.org/abs/1812.00332

[127] T. Veniat and L. Denoyer, "Learning time/memory-efficient deep architectures with budgeted super networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3492–3500.

[128] P. Guo, C.-Y. Lee, and D. Ulbricht, "Learning to branch for multi-task learning," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 3854–3863. [Online]. Available: http://proceedings.mlr.press/v119/guo20e.html

[129] Q. Yao, J. Xu, W.-W. Tu, and Z. Zhu, "Efficient neural architecture search via proximal iterations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[130] X. Yuan, P. H. P. Savarese, and M. Maire, "Growing efficient deep networks by structured continuous sparsification," in *International Conference on Learning Representations*, 2021.

[131] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *International Conference on Learning Representations*, 2019.

[132] M. S. Abdelfattah, A. Mehrotra, Ł. Dudziak, and N. D. Lane, "Zero-cost proxies for lightweight NAS," in *International Conference on Learning Representations*, 2020.

[133] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, "Neural architecture search without training," in *International Conference on Machine Learning*. PMLR, 2021, pp. 7588–7598.

[134] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[135] R. Liu, J. Gao, J. Zhang, D. Meng, and Z. Lin, "Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond," *arXiv preprint arXiv:2101.11517*, 2021.

[136] Y. Yoshihara, M. De Roo, and D. Muller, "Dendritic spine formation and stabilization," *Current opinion in neurobiology*, vol. 19, no. 2, pp. 146–153, 2009.

[137] T. Wei, C. Wang, Y. Rui, and C. W. Chen, "Network morphism," in *International Conference on Machine Learning*. PMLR, 2016, pp. 564–572.

[138] J. Kim, M.-S. Kim, and H. Yoon, "Tweaking deep neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[139] A. Holtmaat and K. Svoboda, "Experience-dependent structural synaptic plasticity in the mammalian brain," *Nature Reviews Neuroscience*, vol. 10, no. 9, pp. 647–658, 2009.

[140] T. E. Faust, G. Gunner, and D. P. Schafer, "Mechanisms governing activity-dependent synaptic pruning in the developing mammalian cns," *Nature Reviews Neuroscience*, pp. 1–17, 2021.

[141] A. Alunni and L. Bally-Cuif, "A comparative view of regenerative neurogenesis in vertebrates," *Development*, vol. 143, no. 5, pp. 741–753, 2016.

[142] U. Evci, B. van Merriënboer, T. Unterthiner, F. Pedregosa, and M. Vladymyrov, "GradMax: Growing neural networks using gradient information," in *10th International Conference on Learning Representations*, 2022.

[143] K. M. Christian, H. Song, and G.-l. Ming, "Functions and dysfunctions of adult hippocampal neurogenesis," *Annual review of neuroscience*, vol. 37, p. 243, 2014.

[144] Q. Liu, L. Wu, and D. Wang, "Splitting steepest descent for growing neural architectures," *Advances in neural information processing systems*, 2019.

[145] C. Dong, L. Liu, Z. Li, and J. Shang, "Towards adaptive residual network training: A neural-ode perspective," in *International Conference on Machine Learning*. PMLR, 2020, pp. 2616–2626.

[146] W. Wen, F. Yan, Y. Chen, and H. Li, "AutoGrow: Automatic Layer Growing in Deep Convolutional Networks," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. 20, pp. 833–841, 2020. [Online]. Available: https://doi.org/10.1145/3394486.3403126

[147] T. Ash, "Dynamic Node Creation in Backpropagation Networks," *Connection Science*, vol. 1, no. 4, pp. 365–375, jan 1989. [Online]. Available: https://sci-hub.do/10.1080/09540098908915647

[148] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems*, 1990, pp. 524–532.

[149] M. Frean, "The upstart algorithm: A method for constructing and training feedforward neural networks," *Neural computation*, vol. 2, no. 2, pp. 198–209, 1990.

[150] M. Lehtokangas, "Modelling with constructive backpropagation," *Neural Networks*, vol. 12, no. 4-5, pp. 707–716, 1999.

[151] L. Ma and K. Khorasani, "A new strategy for adaptively constructing multilayer feedforward neural networks," *Neurocomputing*, vol. 51, pp. 361–385, 2003.

[152] M. M. Islam, M. A. Sattar, M. F. Amin, X. Yao, and K. Murase, "A new constructive algorithm for architectural and functional adaptation of artificial neural networks," *IEEE transactions on systems, man, and cybernetics, part B (cybernetics)*, vol. 39, no. 6, pp. 1590–1605, 2009.

[153] C. Cortes, X. Gonzalvo, V. Kuznetsov, M. Mohri, and S. Yang, "Adanet: Adaptive structural learning of artificial neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 874–883.

[154] S. Liang, R. Sun, J. D. Lee, and R. Srikant, "Adding one neuron can eliminate all bad local minima," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[155] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. Feris, "Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 5334–5343.

[156] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, "Identifying and attacking the saddle point problem in high-dimensional non-convex optimization," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[157] G. G. Turrigiano, "Homeostatic plasticity in neuronal networks: the more things change, the more they stay the same," *Trends in neurosciences*, vol. 22, no. 5, pp. 221–227, 1999.

[158] T. Miconi, A. Rawal, J. Clune, and K. O. Stanley, "Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity," in *International Conference on Learning Representations*, 2019.

[159] J. C. Magee and C. Grienberger, "Synaptic plasticity forms and functions," *Annual review of neuroscience*, vol. 43, pp. 95–117, 2020.

[160] A. D. Milstein, Y. Li, K. C. Bittner, C. Grienberger, I. Soltesz, J. C. Magee, and S. Romani, "Bidirectional synaptic plasticity rapidly modifies hippocampal representations," *eLife*, vol. 10, 2021.

[161] X. Chu, T. Zhou, B. Zhang, and J. Li, "Fair DARTS: Eliminating unfair advantages in differentiable architecture search," in *European Conference on Computer Vision*. Springer, 2020, pp. 465–480.

[162] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[163] L. J. Herstel and C. J. Wierenga, "Network control through coordinated inhibition," *Current Opinion in Neurobiology*, vol. 67, pp. 34–41, 2021.

[164] E. Marder, "Neuromodulation of neuronal circuits: back to the future," *Neuron*, vol. 76, no. 1, pp. 1–11, 2012.

[165] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[166] A. Tavanaei, M. Ghodrati, S. R. Kheradpisheh, T. Masquelier, and A. Maida, "Deep learning in spiking neural networks," *Neural networks*, vol. 111, pp. 47–63, 2019.

[167] D. Kappel, S. Habenschuss, R. Legenstein, and W. Maass, "Network plasticity as bayesian inference," *PLoS Computational Biology*, vol. 11, no. 11, p. e1004485, 2015.

[168] S. Tian, L. Qu, L. Wang, K. Hu, N. Li, and W. Xu, "A neural architecture search based framework for liquid state machine design," *Neurocomputing*, vol. 443, pp. 174–182, 2021.

[169] N. K. Kasabov, "Neucube: A spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data," *Neural Networks*, vol. 52, pp. 62–76, 2014.

[170] E. Olivares, E. J. Izquierdo, and R. D. Beer, "A neuromechanical model of multiple network rhythmic pattern generators for forward locomotion in c. elegans," *Frontiers in Computational Neuroscience*, vol. 15, p. 572339, 2021.

[171] M. Lechner, R. Hasani, A. Amini, T. A. Henzinger, D. Rus, and R. Grosu, "Neural circuit policies enabling auditable autonomy," *Nature Machine Intelligence*, vol. 2, no. 10, pp. 642–652, 2020.

[172] G. F. Miller, P. M. Todd, and S. U. Hegde, "Designing neural networks using genetic algorithms." in *ICGA*, vol. 89, 1989, pp. 379–384.

[173] N. Richards, D. E. Moriarty, and R. Miikkulainen, "Evolving neural networks to play Go," *Applied Intelligence*, vol. 8, no. 1, pp. 85–96, 1998.

[174] J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, J. T. Folsom-Kovarik, and K. O. Stanley, "Picbreeder: A case study in collaborative evolutionary exploration of design space," *Evolutionary computation*, vol. 19, no. 3, pp. 373–403, 2011.

[175] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, "A neuroevolution approach to general atari game playing," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 4, pp. 355–366, 2014.

[176] F. Gruau, "Neural network synthesis using cellular encoding and the genetic algorithm," *PhD thesis, Laboratoire de l'Informatique du Parallilisme, Ecole Normale Supirieure de Lyon*, 1994.

[177] J. F. Miller and D. G. Wilson, "A developmental artificial neural network model for solving multiple problems," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. ACM, 2017, pp. 69–70.

[178] G. S. Hornby and J. B. Pollack, "Creating high-level components with a generative representation for body-brain evolution," *Artificial Life*, vol. 8(3), 2002.

[179] M. Huisman, J. N. Van Rijn, and A. Plaat, "A survey of deep meta-learning," *Artificial Intelligence Review*, vol. 54, no. 6, pp. 4483–4541, 2021.

[180] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *International Conference on Learning Representations*, 2017.

[181] Y. Chen, X. Song, C. Lee, Z. Wang, Q. Zhang, D. Dohan, K. Kawakami, G. Kochanski, A. Doucet, M. Ranzato *et al.*, "Towards learning universal hyperparameter optimizers with transformers," *arXiv preprint arXiv:2205.13320*, 2022.

[182] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.

[183] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[184] S. Niu, J. Wu, G. Xu, Y. Zhang, Y. Guo, P. Zhao, P. Wang, and M. Tan, "Adaxpert: Adapting neural architecture for growing data," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 18–24 Jul 2021, pp. 8184–8194. [Online]. Available: http://proceedings.mlr.press/v139/niu21a.html

[185] G. S. Hornby and J. B. Pollack, "Body-brain co-evolution using L-systems as a generative encoding," in *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, 2001, pp. 868–875.

[186] Y. Fu, Z. Yu, Y. Zhang, and Y. Lin, "Auto-agent-distiller: Towards efficient deep reinforcement learning agents via neural architecture search," *arXiv preprint arXiv:2012.13091*, 2020.

[187] Y. Miao, X. Song, D. Peng, S. Yue, E. Brevdo, and A. Faust, "RL-DARTS: Differentiable architecture search for reinforcement learning," *arXiv preprint arXiv:2106.02229*, 2021.

[188] M. Botvinick, J. X. Wang, W. Dabney, K. J. Miller, and Z. Kurth-Nelson, "Deep reinforcement learning and its neuroscientific implications," *Neuron*, 2020.

[189] A. Banerjee, R. V. Rikhye, and A. Marblestone, "Reinforcement-guided learning in frontal neocortex: emerging computational concepts," *Current Opinion in Behavioral Sciences*, vol. 38, pp. 133–140, 2021.

[190] F. Huang and H. Huang, "Biadam: Fast adaptive bilevel optimization methods," *arXiv preprint arXiv:2106.11396*, 2021.

[191] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," *arXiv preprint arXiv:2104.13478*, 2021.

[192] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" *arXiv preprint arXiv:2003.03033*, 2020.

[193] Y. Bian, Q. Song, M. Du, J. Yao, H. Chen, and X. Hu, "Sub-architecture ensemble pruning in neural architecture search," *arXiv preprint arXiv:1910.00370*, 2019.

[194] N. Sinha and K.-W. Chen, "Evolving neural architecture using one shot model," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2021, pp. 910–918.

[195] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten ZIP code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[196] T. Cohen and M. Welling, "Group equivariant convolutional networks," in *International Conference on Machine Learning*. PMLR, 2016, pp. 2990–2999.

[197] S. Ravanbakhsh, J. Schneider, and B. Poczos, "Equivariance through parameter-sharing," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2892–2901.

[198] R. Kondor and S. Trivedi, "On the generalization of equivariance and convolution in neural networks to the action of compact groups," in *International Conference on Machine Learning*. PMLR, 2018, pp. 2747–2755.

[199] M. Weiler, P. Forré, E. Verlinde, and M. Welling, "Coordinate independent convolutional networks - isometry and gauge equivariant convolutions on riemannian manifolds," *arXiv preprint arXiv:2106.06020*, 2021.

[200] M. Weiler and G. Cesa, "General E(2)-equivariant steerable CNNs," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[201] R. Wang, R. Walters, and R. Yu, "Approximately equivariant networks for imperfectly symmetric dynamics," in *International Conference on Machine Learning*. PMLR, 2022.

[202] M. Finzi, G. Benton, and A. G. Wilson, "Residual pathway priors for soft equivariance constraints," *Advances in Neural Information Processing Systems*, vol. 34, pp. 30 037–30 049, 2021.

[203] A. Zhou, T. Knowles, and C. Finn, "Meta-learning symmetries by reparameterization," in *International Conference on Learning Representations*, 2020.

[204] R. A. Yeh, Y.-T. Hu, M. Hasegawa-Johnson, and A. Schwing, "Equivariance discovery by learned parameter-sharing," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022, pp. 1527–1545.

[205] S. Basu, A. Magesh, H. Yadav, and L. R. Varshney, "Autoequivariant network search via group decomposition," *arXiv preprint arXiv:2104.04848*, 2021.

[206] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2902–2911.

[207] Z. Lu, I. Whalen, V. Boddeti, Y. Dhebar, K. Deb, E. Goodman, and W. Banzhaf, "NSGA-Net: neural architecture search using multi-objective genetic algorithm," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019, pp. 419–427.

[208] D. W. Romero and S. Lohit, "Learning partial equivariances from data," *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[209] T. F. van der Ouderaa, D. W. Romero, and M. van der Wilk, "Relaxing equivariance constraints with non-stationary continuous filters," *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[210] D. Agrawal and J. Ostrowski, "A classification of G-invariant shallow neural networks," *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[211] I. N. Herstein, *Topics in algebra*. John Wiley & Sons, 2006.

[212] A. Falanti, E. Lomurno, S. Samele, D. Ardagna, M. Matteucci *et al.*, "POPNASv2: An efficient multi-objective neural architecture search technique," in *International Joint Conference on Neural Networks*, 2022.

[213] G. Elsayed, P. Ramachandran, J. Shlens, and S. Kornblith, "Revisiting spatial invariance with low-rank local connectivity," in *International Conference on Machine Learning*. PMLR, 2020, pp. 2868–2879.

[214] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," *Advances in Neural Information Processing Systems*, vol. 29, 2016.

[215] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *International Conference on Machine Learning*. PMLR, 2007, pp. 473–480.

[216] H. W. Leung and J. Bovy, "Deep learning of multi-element abundances from high-resolution spectroscopic data," *Monthly Notices of the Royal Astronomical Society*, vol. 483, no. 3, pp. 3255–3277, 2019.

[217] N. C. Codella, D. Gutman, M. E. Celebi, B. Helba, M. A. Marchetti, S. W. Dusza, A. Kalloo, K. Liopyris, N. Mishra, H. Kittler *et al.*, "Skin lesion analysis toward melanoma detection: A challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), hosted by the International Skin Imaging Collaboration (ISIC)," in *International Symposium on Biomedical Imaging*. IEEE, 2018, pp. 168–172.

[218] P. Tschandl, C. Rosendahl, and H. Kittler, "The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions," *Scientific Data*, vol. 5, no. 1, pp. 1–9, 2018.

[219] M. Combalia, N. C. Codella, V. Rotemberg, B. Helba, V. Vilaplana, O. Reiter, C. Carrera, A. Barreiro, A. C. Halpern, S. Puig *et al.*, "BCN20000: Dermoscopic lesions in the wild," *arXiv preprint arXiv:1908.02288*, 2019.

[220] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *International Conference on Machine Learning*. PMLR, 2015, pp. 2113–2122.

[221] J. Lichtarge, C. Alberti, and S. Kumar, "Simple and effective gradient-based tuning of sequence-to-sequence models," *International Conference on Automated Machine Learning*, 2022.

[222] L. Xie, X. Chen, K. Bi, L. Wei, Y. Xu, L. Wang, Z. Chen, A. Xiao, J. Chang, X. Zhang *et al.*, "Weight-sharing neural architecture search: A battle to shrink the optimization gap," *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–37, 2021.

[223] Y. Li, P. Zhao, G. Yuan, X. Lin, Y. Wang, and X. Chen, "Pruning-as-search: Efficient neural architecture search via channel pruning and structural reparameterization," in *International Joint Conference on Artificial Intelligence*, 2022.

[224] C. White, M. Khodak, R. Tu, S. Shah, S. Bubeck, and D. Dey, "A deeper look at zero-cost proxies for lightweight NAS," in *ICLR Blog Track*, 2022. [Online]. Available: https://iclr-blog-track.github.io/2022/03/25/zero-cost-proxies/

[225] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4780–4789.

[226] Y. Li, Y. Zhou, Y. Wang, and Z. Tang, "PD-DARTS: Progressive discretization differentiable architecture search," in *International Conference on Pattern Recognition and Artificial Intelligence*. Springer, 2020, pp. 306–311.

[227] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1294–1303.

[228] R. Wang, M. Cheng, X. Chen, X. Tang, and C.-J. Hsieh, "Rethinking architecture selection in differentiable NAS," in *International Conference on Learning Representations*, 2020.

[229] X. Chu, X. Wang, B. Zhang, S. Lu, X. Wei, and J. Yan, "DARTS-: robustly stepping out of performance collapse without indicators," *arXiv preprint arXiv:2009.01027*, 2020.

[230] P. Zhou, C. Xiong, R. Socher, and S. C. Hoi, "Theory-inspired path-regularized differential network architecture search," *arXiv preprint arXiv:2006.16537*, 2020.

[231] T. E. Arber Zela, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter, "Understanding and robustifying differentiable architecture search," in *International Conference on Learning Representations*, vol. 3, 2020, p. 7.

[232] H. Liang, S. Zhang, J. Sun, X. He, W. Huang, K. Zhuang, and Z. Li, "DARTS+: Improved differentiable architecture search with early stopping," *arXiv preprint arXiv:1909.06035*, 2019.

[233] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[234] J. Martens, *Second-order optimization for neural networks*. University of Toronto (Canada), 2016.

[235] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[236] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[237] M. Tu, V. Berisha, Y. Cao, and J.-s. Seo, "Reducing the model order of deep neural networks using information theory," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2016, pp. 93–98.

[238] L. Theis, I. Korshunova, A. Tejani, and F. Huszár, "Faster gaze prediction with dense networks and fisher pruning," *arXiv preprint arXiv:1801.05787*, 2018.

[239] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends R in Machine Learning*, vol. 3, pp. 1–122, 2010.

[240] F. Kiaee, C. Gagné, and M. Abbasi, "Alternating direction method of multipliers for sparse convolutional neural networks," *arXiv preprint arXiv:1611.01590*, 2016.

[241] T. Zhang, S. Ye, K. Zhang, X. Ma, N. Liu, L. Zhang, J. Tang, K. Ma, X. Lin, M. Fardad *et al.*, "Structadmm: A systematic, high-efficiency framework of structured weight pruning for dnns," *arXiv preprint arXiv:1807.11091*, 2018.

[242] S. Ye, T. Zhang, K. Zhang, J. Li, K. Xu, Y. Yang, F. Yu, J. Tang, M. Fardad, S. Liu *et al.*, "Progressive weight pruning of deep neural networks using admm," *arXiv preprint arXiv:1810.07378*, 2018.

[243] X. Ma, G. Yuan, S. Lin, Z. Li, H. Sun, and Y. Wang, "Resnet can be pruned 60×: Introducing network purification and unused path removal (p-rm) after weight pruning," in *2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 2019, pp. 1–2.

[244] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 19–34.

[245] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4095–4104.

[246] X. Dong and Y. Yang, "Searching for a robust neural architecture in four gpu hours," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1761–1770.

[247] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 and cifar-100 datasets," *URl: https://www. cs. toronto. edu/kriz/cifar. html*, vol. 6, no. 1, p. 1, 2009.

[248] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of english: the penn treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[249] Y. Bengio, N. Roux, P. Vincent, O. Delalleau, and P. Marcotte, "Convex neural networks," in *Advances in Neural Information Processing Systems*, vol. 18, 2005.

[250] L. Wu, D. Wang, and Q. Liu, "Splitting steepest descent for growing neural architectures," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[251] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," in *2nd International Conference on Learning Representations*, 2014.

[252] H. Daneshmand, A. Joudaki, and F. Bach, "Batch normalization orthogonalizes representations in deep random networks," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 4896–4906.

[253] C. Lyle, M. Rowland, and W. Dabney, "Understanding and preventing capacity loss in reinforcement learning," in *10th International Conference on Learning Representations*, 2022.

[254] L. Jing, P. Vincent, Y. LeCun, and Y. Tian, "Understanding dimensional collapse in contrastive self-supervised learning," in *10th International Conference on Learning Representations*, 2022.

[255] T. Chen, I. Goodfellow, and J. Shlens, "Net2Net: Accelerating learning via knowledge transfer," in *4th International Conference on Learning Representations*, 2016.

[256] Y. Kilcher, G. Bécigneul, and T. Hofmann, "Escaping flat areas via function-preserving structural network modifications," *openreview.net*, 2018.

[257] A. Kumar, R. Agarwal, D. Ghosh, and S. Levine, "Implicit under-parameterization inhibits data-efficient deep reinforcement learning," in *9th International Conference on Learning Representations*, 2021.

[258] D. P. Bertsekas, "Nonlinear programming," *Journal of the Operational Research Society*, vol. 48, no. 3, pp. 334–334, 1997.

[259] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[260] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE signal processing magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[261] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations*, 2015.

[262] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[263] D. P. Kingma and J. Ba, "ADAM: A method for stochastic optimization," in *3rd International Conference on Learning Representations*, 2015.

[264] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9. PMLR, 2010, pp. 249–256.

[265] D. Bertoin, J. Bolte, S. Gerchinovitz, and E. Pauwels, "Numerical influence of ReLU'(0) on back-propagation," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 468–479.

[266] A. Lacoste, A. Luccioni, V. Schmidt, and T. Dandres, "Quantifying the carbon emissions of machine learning," *arXiv preprint arXiv:1910.09700*, 2019.

[267] L. Lannelongue, J. Grealey, and M. Inouye, "Green algorithms: Quantifying the carbon footprint of computation," *Advanced Science*, vol. 8, no. 12, p. 2100707, 2021.

[268] M. Hein, M. Andriushchenko, and J. Bitterwolf, "Why relu networks yield high-confidence predictions far away from the training data and how to mitigate the problem," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 41–50.

[269] G. Sokar, R. Agarwal, P. S. Castro, and U. Evci, "The dormant neuron phenomenon in deep reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2023.

[270] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[271] J. Howard, "Imagenette," https://github.com/fastai/imagenette.

[272] Y. Lee, A. S. Chen, F. Tajwar, A. Kumar, H. Yao, P. Liang, and C. Finn, "Surgical fine-tuning improves adaptation to distribution shifts," *11th International Conference on Learning Representations*, 2023.

[273] K. Azarian, Y. Bhalgat, J. Lee, and T. Blankevoort, "Learned threshold pruning," *arXiv preprint arXiv:2003.00075*, 2020.

[274] R. T. Lange, T. Schaul, Y. Chen, T. Zahavy, V. Dallibard, C. Lu, S. Singh, and S. Flennerhag, "Discovering evolution strategies via meta-black-box optimization," *11th International Conference on Learning Representations*, 2023.

[275] E. Real, C. Liang, D. So, and Q. Le, "AutoML-Zero: Evolving machine learning algorithms from scratch," in *International Conference on Machine Learning*. PMLR, 2020.

[276] S. Dohare, R. S. Sutton, and A. R. Mahmood, "Continual backprop: Stochastic gradient descent with persistent randomness," *arXiv preprint arXiv:2108.06325*, 2021.

[277] E. Nikishin, M. Schwarzer, P. D'Oro, P.-L. Bacon, and A. Courville, "The primacy bias in deep reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2022, pp. 16 828–16 847.

[278] S. Qiao, Z. Lin, J. Zhang, and A. L. Yuille, "Neural rejuvenation: Improving deep network training by enhancing computational resource utilization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 61–71.

[279] X. Dai, H. Yin, and N. K. Jha, "Incremental learning using a grow-and-prune paradigm with efficient neural networks," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 752–762, 2020.

[280] F. Tung, S. Muralidharan, and G. Mori, "Fine-pruning: Joint fine-tuning and compression of a convolutional network with bayesian optimization," in *British Machine Vision Conference (BMVC)*, 2017.

[281] M. Santacroce, Z. Wen, Y. Shen, and Y. Li, "What matters in the structured pruning of generative language models?" *arXiv preprint arXiv:2302.03773*, 2023.

[282] C. Zhao, Y. Zhang, and B. Ni, "Exploiting channel similarity for network pruning," *IEEE Transactions on Circuits and Systems for Video Technology*, 2023.

[283] P. P. Liang, C. Wu, L.-P. Morency, and R. Salakhutdinov, "Towards understanding and mitigating social biases in language models," in *International Conference on Machine Learning*. PMLR, 2021, pp. 6565–6576.