When, where, and how to add new neurons to ANNs

Anonymous[1]

[1]Anonymous Institution

Abstract 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. [1]

1 Introduction

Training deep artificial neural networks (ANNs) usually involves pre-selecting a static architecture and then optimizing the parameters of that architecture for a given dataset and task. However, the field of *structural learning* aims to optimize both the architecture and its parameters during the learning process through techniques such as neural architecture search (NAS) or pruning. 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. 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. Neurogenesis is the form of structural learning that adds new neurons or channels, complementary to structural pruning often used by channel-searching NAS techniques. Our focus in this paper is neurogenesis, which is a naturally difficult 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.
- 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.

[1]Our code is available anonymously here: `https://github.com/neurogenesisauthors/Neurogenesis`

 © 2022 the authors, released under CC BY 4.0

2 Background

2.1 Problem Statement and Notations

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

$$\arg\min_{f} \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}\sim D}\, L(f(\boldsymbol{x}), \boldsymbol{y}) \tag{1}$$

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 lth 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 lth 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 (1) defines the neurogenesis optimization. We restrict the search space of where to add new neurons to within existing layers.

2.2 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 a 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 Ash (1989), Bengio et al. (2005), and Liu et al. (2019) 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 Wu et al. (2020) generates candidate neurons that either split existing neurons with noise or are completely new and selects those with the highest gradient norm. Du et al. (2019) also create noisy copies, selecting from high-saliency neurons. Both GradMax from Evci et al. (2022) and NeST from Dai et al. (2019) 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, Saxe et al. (2014) 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. Daneshmand et al. (2021) finds pre-activation orthogonality

Algorithm 1 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
```

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. Mellor et al. (2021) proposes a NAS heuristic based on post-activation distances between samples, computationally similar to the orthogonality gap metric by Daneshmand et al. (2021). Maintaining orthogonality of neuron activations in static networks has especially been studied in reinforcement learning (Lyle et al., 2022) and self-supervised learning (Jing et al., 2022), 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 Evci et al. (2022) and Islam et al. (2009), or may change the function and current performance. Function-preserving neurogenesis is a form of network morphism (Wei et al., 2016; Chen et al., 2016), maintaining the location in the newly expanded loss landscape but changing the direction of the descent path into the new dimensions. Evci et al. (2022) selects the steepest direction by maximising the gradient norm. Function-altering neurons jump from the descent path: Kilcher et al. (2018), Liu et al. (2019), and Wu et al. (2020) escape convergence at saddle points, while Dai et al. (2019) 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.

3 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 1. 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 Appendix A.4. Our strategies with their triggers and initializations are summarized in Table 1. We define triggers in Section 3.1 and initializations in Section 3.2.

3.1 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.

3.1.1 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

Table 1: Specification of initialization and trigger pairs used for each strategy.

Strategy	Trigger		Initialization	
NORTH-Select	T_{act}	sec. 3.1.1, eq. (3)	Select	sec. 3.2.1
NORTH-Pre	T_{act}	sec. 3.1.1, eq. (3)	Pre-activation	sec. 3.2.1, eq. (7)
NORTH-Random	T_{act}	sec. 3.1.1, eq. (3)	RandomInit	sec. 3.2.1
NORTH-Weight	T_{weight}	sec. 3.1.2, eq. (5)	Weight	sec. 3.2.2
GradMax	T_{grad}	sec. 3.1.3, eq. (6)	GradMax	sec. 3.2.3, Evci et al. (2022)
Firefly	T_{grad}	sec. 3.1.3, eq. (6)	Firefly	sec. 3.2.3, Wu et al. (2020)
NeST	T_{grad}	sec. 3.1.3, eq. (6)	NeST	sec. 3.2.3, Dai et al. (2019)

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 adapt a metric from Lyle et al. (2022) based on the count of singular values of the post-activations above a small threshold. For layer l and n samples generating the post-activation $\boldsymbol{H}_l$, the effective dimension metric may be estimated by

$$\phi_a^{ED}(f,l) = \frac{1}{M_l}\left|\left\{\sigma \in \text{SVD}\left(\frac{1}{\sqrt{n}}\boldsymbol{H}_l\right) \middle| \sigma > \epsilon\right\}\right|, \tag{2}$$

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 hard constraint 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. 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 the network. Similar to Lyle et al. (2022), we aim to maintain this initial metric value over training, even as the layer grows. We multiply this 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:

$$T_{act}(f,\phi_a,l) = \min\left(0, \lfloor M_l\left(\phi_a(f,l) - \gamma_a\phi_a(f_0,l)\right)\rfloor\right), \tag{3}$$

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

3.1.2 Weight-based. We include weight matrix orthogonality as a comparison to methods based on the post-activations. The trigger, T_{weight}, is computed as in equations (2)-(3) but with $\boldsymbol{W}_l$ instead of $\boldsymbol{H}_l$:

$$\phi_w^{ED}(f,l) = \frac{1}{M_l}\left|\left\{\sigma \in \text{SVD}\left(\frac{1}{\sqrt{n}}\boldsymbol{W}_l\right) \middle| \sigma > \epsilon\right\}\right|, \tag{4}$$

$$T_{weight}(f,\phi_w,l) = \min\left(0, \lfloor M_l\left(\phi_w(f,l) - \gamma_w\phi_w(f_0,l)\right)\rfloor\right). \tag{5}$$

3.1.3 Gradient-based. In order to study gradient-based neurogenesis initializations such as Dai et al. (2019), Wu et al. (2020), and Evci et al. (2022) in the context of dynamic neurogenesis, we propose a trigger based on the auxiliary gradient. As shown by Evci et al. (2022), the maximum increase in gradient norm by adding k neurons to the lth layer is the sum of the largest k singular values of the auxiliary gradient matrix $\frac{\partial L}{\partial z_{l+1}}\boldsymbol{h}_{l-1}^{\intercal}$. 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:

$$T_{grad}(f,L,l) = \left|\left\{\sigma \in \text{SVD}\left(\frac{\partial L}{\partial \boldsymbol{Z}_{l+1}}\boldsymbol{H}_{l-1}^{\intercal}\right) \middle| \sigma > \sum_{m=1}^{M_l}\left\|\frac{\partial L}{\partial \boldsymbol{w}_m^{\text{in}}}\right\|_F + \left\|\frac{\partial L}{\partial \boldsymbol{w}_m^{\text{out}}}\right\|_F\right\}\right|, \tag{6}$$

where $\boldsymbol{w}_m^{\text{in}}$ is the mth row of $\boldsymbol{W}_l$ and $\boldsymbol{w}_m^{\text{out}}$ is the mth column of $\boldsymbol{W}_{l+1}$, respectively representing the fan-in and fan-out weights of the mth neuron in layer l. We intuit this threshold will be harder to surpass as the layer grows, thus leading to convergence in layer width.

3.2 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 follow a similar notion as in Evci et al. (2022) by adding 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.

3.2.1 Activation-based.

We present multiple approaches for initializing neurons towards orthogonal post-activations. Due to the nonlinearity of a neuron, we do not have a closed-form solution for a set of fan-in weights yielding an orthogonal post-activation. The following approaches generate candidate neurons and select those that independently maximise the orthogonality metric with the existing neurons in that layer. We initialize fan-out weights to 0. See Appendix A.2 for more details.

- 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

$$\boldsymbol{w} = (\boldsymbol{H}_{l-1}^{\top})^{-1} \boldsymbol{V}_{\boldsymbol{Z}_l}'^{\top} \boldsymbol{a}, \tag{7}$$

  where $(\boldsymbol{H}_{l-1}^{\top})^{-1}$ 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.

3.2.2 Weight-based.

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

3.2.3 Gradient-based.

We reimplement the neurogenesis initialization portions of NeST from Dai et al. (2019), Firefly from Wu et al. (2020), and GradMax from Evci et al. (2022) within our framework. We summarize the strategies and note any differences from the original works in Appendix A.3.

4 Experiments

We study the impact of the different trigger and initialization methods detailed in Table 1 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 tasks: MLPs (Rumelhart et al., 1986) on generated toy datasets in Section 4.1, MLPs on MNIST (Deng, 2012) in Section 4.2, and VGG-11 (Simonyan and Zisserman, 2015) on CIFAR10 in Section 4.3. For all experiments, plots show mean and standard deviation across 5 seeded trials in the form of error bars, clouds, or ellipses. We include further experiment details in Appendix B.1 and a study on hyperparameters in B.2. We only compare our results

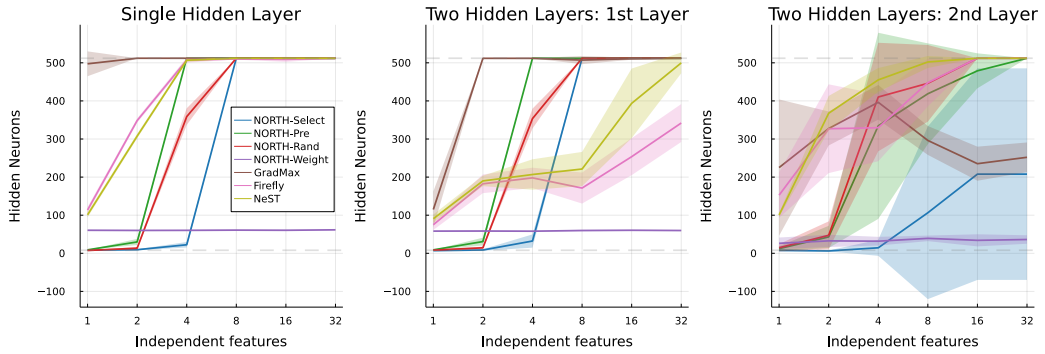

Figure 1: Hidden neurons upon convergence on for MLPs on generated toy data versus the number of independent features in a 1 (left) and 2 (middle, right) hidden layer MLPs.

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.

4.1 Generated Data

We first analyze all strategies on supervised learning for toy generated binary classification datasets, described in Appendix B.1.1. 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. We train MLPs with 1 or 2 hidden layers using the different neurogenesis strategies in all layers, shown in Figure 1. 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.

4.2 MLP MNIST

We continue our study on neurogenesis in dense MLP layers on the task of MNIST classification using networks with vectorized image input, 2 hidden layers which grow using neurogenesis, and 10 output neurons. We test dynamic strategies as well as isolate the initializations in preset growth schedules, which are detailed in Appendix B.1.2. We compare test accuracy and training time against final network size or schedule in Figure 2, detailing the layer widths over training for dynamic schedules in Figure 3.

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

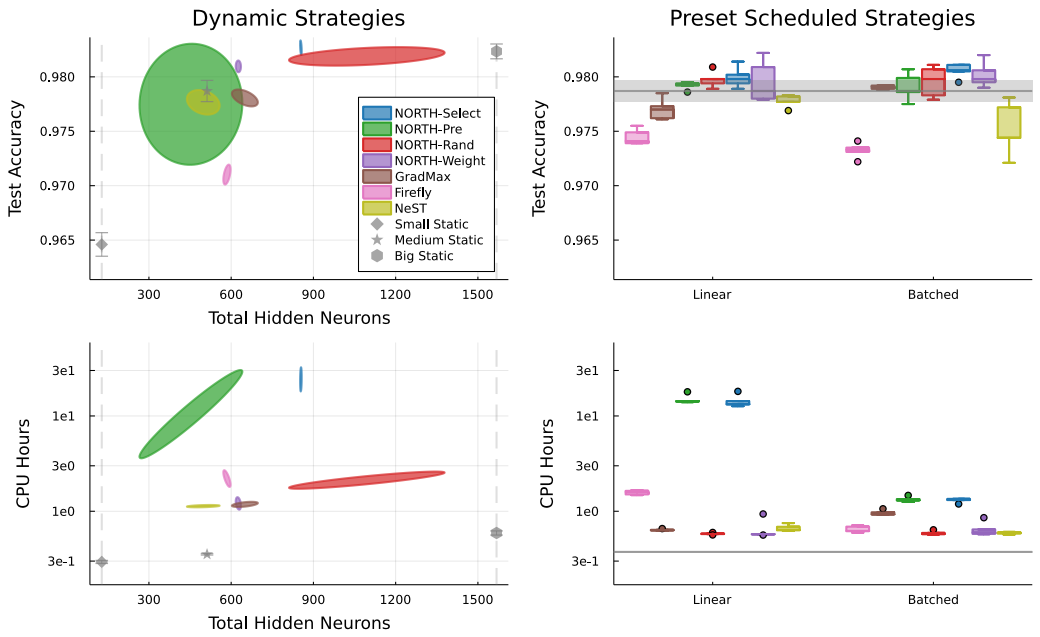

Figure 2: Dynamic (left) and preset scheduled (right) comparisons of network performance across neurogenesis strategies for MLPs on MNIST, evaluated by test accuracy (top) and training time (bottom).

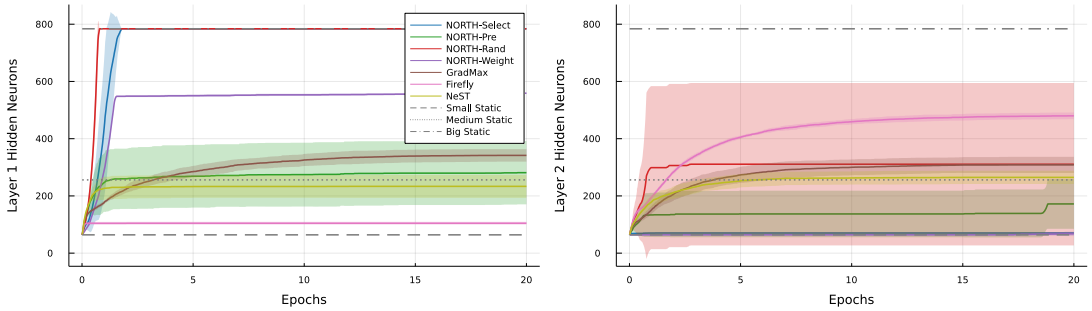

Figure 3: Layer width for MLPs on MNIST over training for dynamic strategies.

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 large SVD calculations used by NORTH* strategies hinder efficiency.

From Figure 3, 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

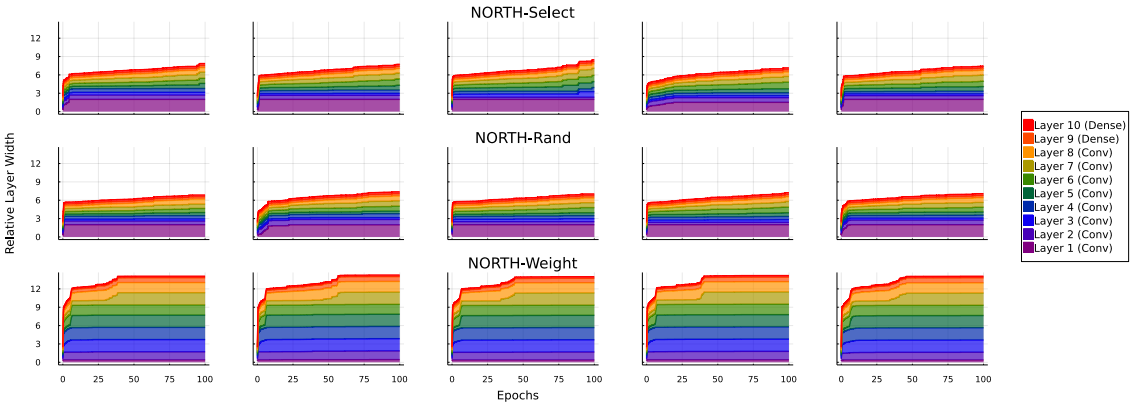

Figure 5: Layer widths for VGG-11 on CIFAR10, scaled relative to standard VGG-11 widths, over the course of training for NORTH-Select (top), NORTH-Rand (middle), and NORTH-Weight (bottom), across each seed 1-5 in order. The stacked area represents the size of each layer as the difference between lines.

approaches of NORTH-Select and NORTH-Random, as well as NORTH-Weight which has a similar premise towards orthogonal pre-activations.

4.3 VGG-11 on CIFAR10

We extend neurogenesis to deep convolutional networks and test them on CIFAR10 classification with a VGG11 backbone. Due to restricted time and resources, we only test the dynamic NORTH* methods applicable to convolutions against the non-growing baselines. Further details are in Appendix B.1.3.

Test accuracy of the different methods is compared against network size in Figure 4. Our NORTH* methods find networks significantly smaller than standard VGG-11, here represented by Medium Static, while being competitive in accuracy. Importantly, the activation-based strategies NORTH-Select and NORTH-Rand find networks over 6 times smaller than VGG-11 but with similar performance. This demonstrates that neurogenesis can be used to grow efficient networks, similar to pruning trained networks.

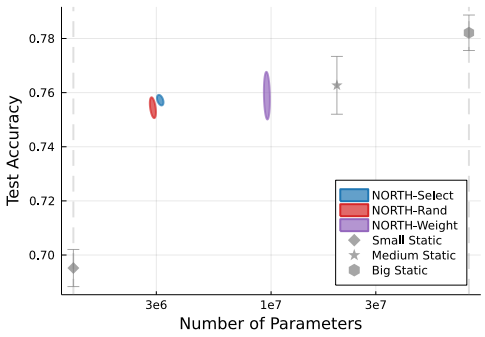

Figure 4: Comparisons of test accuracy vs. network size across neurogenesis strategies for VGG-11 on CIFAR10.

Layer width over training is presented in Figure 5, showing how different NORTH* methods respond to the "Where" and "When" questions. Activation-based strategies quickly grow the first layer to the maximum size, similarly to the MLP case. NORTH-Weight grows later convolutional layers to maximum size. Both activation and weight-based strategies keep dense layers relatively small and grow convolutional layers at a decaying rate throughout training, although there are moments of heightened neurogenesis, such as in seed 3 for NORTH-Select and in layer 7 across all seeds in NORTH-Weight. This shows that triggers respond dynamically to training, trigger neurogenesis at different times and in different layers throughout training.

5 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 and the size of the activation buffer are hyperparameters of NORTH* which can be reduced for efficiency or increased for performance.

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 pruning, as future work, in addition to comparing neurogenesis to 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 direct or incidental potential negative societal impacts beyond those associated with general-purpose machine learning methodologies.

6 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 with established pruning methods. Neurogenesis can grow network architectures that dynamically respond to learning.

7 Reproducibility Checklist

1. For all authors…

(a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] Our title, abstract, and final paragraph of our introduction summarize the claims that are made and supported in the remainder of the paper.

(b) Did you describe the limitations of your work? [Yes] We restrict our search space to adding neurons to existing dense or convolutional layers in supervised learning tasks, although we discuss possible extensions. We discuss our use of maximum layer sizes and hyperparameters to potentiate neurogenesis without pruning to converge to efficient network sizes.

(c) Did you discuss any potential negative societal impacts of your work? [No] We do not identify any pertinent potential negative societal impacts beyond those related to general-purpose machine learning research.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [No] We support these elevated ethics review guidelines and conform to section and most of what is thus far applicable of section 3. However, we could not completely conform to the accessibility guidelines (3.1) with the strict page-limit leading us to use smaller fonts in our figures and the large number of experiments per plot partially limiting our ability to use distinctions beyond color.

2. If you are including theoretical results…

(a) Did you state the full set of assumptions of all theoretical results? [N/A] No theoretical results included.

(b) Did you include complete proofs of all theoretical results? [N/A] No theoretical results included.

3. If you ran experiments…

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] See `https://github.com/neurogenesisauthors/Neurogenesis`.

(b) Did you include the raw results of running the given instructions on the given code and data? [Yes] They are stored in `https://github.com/neurogenesisauthors/Neurogenesis/tree/master/outputs`.

(c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [Yes] The script is found at `https://github.com/neurogenesisauthors/Neurogenesis/blob/master/exp/plot.jl`.

(d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes] Our code was tested and easy to follow.

(e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] Full experiment details are provided in Appendix B.1.

(f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes] We compare results across neurogenesis strategies and baselines within our own implementation using the same dataset, code, and hyperparameters

for each experiment. We acknowledge that our results of standard VGG11 on CIFAR10 does not meet SOTA due to not using data augmentation, dropout, and extensive hyperparameter tuning.

(g) Did you run ablation studies to assess the impact of different components of your approach? [Yes] We study preset schedules to isolate initializations and random initialization to isolate trigger.

(h) Did you use the same evaluation protocol for the methods being compared? [Yes] We only compare results within our own implementation, and all compared results only differ in neurogenesis strategy.

(i) Did you compare performance over time? [Yes] All main experiments that compare accuracy also compare CPU/GPU time.

(j) Did you perform multiple runs of your experiments and report random seeds? [Yes] All experiments were run with seeds 1-5, as described in the Appendix.

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] All results use error bars/ellipses/ribbons that represent standard deviation across 5 trials.

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes] We study our methods in-depth for a toy generated dataset in 4.1 and the well-studied supervised learning task of MNIST in 4.2

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] These details are included in Appendix B.1.

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [Yes] Experiments for sweeping hyperparameters are included in Appendix B.2, and the rest were selected during development or taken from standard practices or default settings.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets…

(a) If your work uses existing assets, did you cite the creators? [Yes] All existing assets are publicly available datasets, code, and models, and are cited in 4.

(b) Did you mention the license of the assets? [N/A] Our experiments were conducted on publicly available or generated datasets, code, and models.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] The only new asset of generated toy data is described in Appendix B.1 and included in our code.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] Our experiments were conducted on publicly available datasets, code, and models.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] Our experiments were conducted on publicly available datasets, code, and models without personally identifiable information or offensive content.

5. If you used crowdsourcing or conducted research with human subjects…

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] No crowdsourcing or human subjects.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] No crowdsourcing or human subjects.

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] No crowdsourcing or human subjects.

References

Ash, T. (1989). Dynamic Node Creation in Backpropagation Networks. *Connection Science*, 1(4):365–375.

Bengio, Y., Roux, N., Vincent, P., Delalleau, O., and Marcotte, P. (2005). Convex neural networks. In *Advances in Neural Information Processing Systems*.

Bertsekas, D. P. (1997). Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334.

Chen, T., Goodfellow, I., and Shlens, J. (2016). Net2net: Accelerating learning via knowledge transfer. In *International Conference on Learning Representations*.

Dai, X., Yin, H., and Jha, N. K. (2019). NeST: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497.

Daneshmand, H., Joudaki, A., and Bach, F. (2021). Batch normalization orthogonalizes representations in deep random networks. In *Advances in Neural Information Processing Systems*.

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research. *IEEE signal processing magazine*, 29(6):141–142.

Du, X., Li, Z., Ma, Y., and Cao, Y. (2019). Efficient network construction through structural plasticity. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(3):453–464.

Evci, U., van Merriënboer, B., Unterthiner, T., Pedregosa, F., and Vladymyrov, M. (2022). Gradmax: Growing neural networks using gradient information. In *International Conference on Learning Representations*.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pages 249–256. PMLR.

Islam, M. M., Sattar, M. A., Amin, M. F., Yao, X., and Murase, K. (2009). A new constructive algorithm for architectural and functional adaptation of artificial neural networks. *IEEE transactions on systems, man, and cybernetics, part B (cybernetics)*, 39(6):1590–1605.

Jing, L., Vincent, P., LeCun, Y., and Tian, Y. (2022). Understanding dimensional collapse in contrastive self-supervised learning. In *International Conference on Learning Representations*.

Kilcher, Y., Bécigneul, G., and Hofmann, T. (2018). Escaping flat areas via function-preserving structural network modifications.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lacoste, A., Luccioni, A., Schmidt, V., and Dandres, T. (2019). Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*.

Lannelongue, L., Grealey, J., and Inouye, M. (2021). Green algorithms: Quantifying the carbon footprint of computation. *Advanced Science*, 8(12):2100707.

Liu, Q., Wu, L., and Wang, D. (2019). Splitting steepest descent for growing neural architectures. In *Advances in Neural Information Processing Systems*.

Lyle, C., Rowland, M., and Dabney, W. (2022). Understanding and preventing capacity loss in reinforcement learning. In *International Conference on Learning Representations*.

Mellor, J., Turner, J., Storkey, A., and Crowley, E. J. (2021). Neural architecture search without training. In *International Conference on Machine Learning*, pages 7588–7598. PMLR.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2014). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.

Wei, T., Wang, C., Rui, Y., and Chen, C. W. (2016). Network morphism. In *International Conference on Machine Learning*, pages 564–572. PMLR.

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