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

Kaitlin Maile[1] **Emmanuel Rachelson**[2] **Hervé Luga**[1] **Dennis G. Wilson**[2]

[1]IRIT, University of Toulouse, Toulouse, France
[2]ISAE-SUPAERO, University of Toulouse, Toulouse, France

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 (Goodfellow et al., 2016). 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) (Elsken et al., 2019) or pruning (Deng et al., 2020). 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 Li et al. (2017), Wan et al. (2020) and to network deepening such as Wen et al. (2020). Our focus in this paper 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.

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

- 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.

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 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 (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 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); Wu 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.

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

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 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); 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); Wu et al. (2019, 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

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

Strategy	Trigger		Initialization	
NORTH-Select	T_{act}	eq. (3)	Select	sec. 3.2
NORTH-Pre	T_{act}	eq. (3)	Pre-activation	eq. (7)
NORTH-Random	T_{act}	eq. (3)	RandomInit	sec. 3.2
NORTH-Weight	T_{weight}	eq. (5)	Weight	eq. (8)
GradMax	T_{grad}	eq. (6)	GradMax	sec. A.3, Evci et al. (2022)
Firefly	T_{grad}	eq. (6)	Firefly	sec. A.3, Wu et al. (2020)
NeST	T_{grad}	eq. (6)	NeST	sec. A.3, Dai et al. (2019)

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 (Kumar et al., 2021; Lyle et al., 2022). 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 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:

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

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

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

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

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

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 l^{th} 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 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.

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 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 Saxe et al. (2014). 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. 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

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

  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.

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 $\boldsymbol{W}_l$:

$$\boldsymbol{w} = \text{proj}_{\ker W_l}(\boldsymbol{w}_i), \tag{8}$$

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.

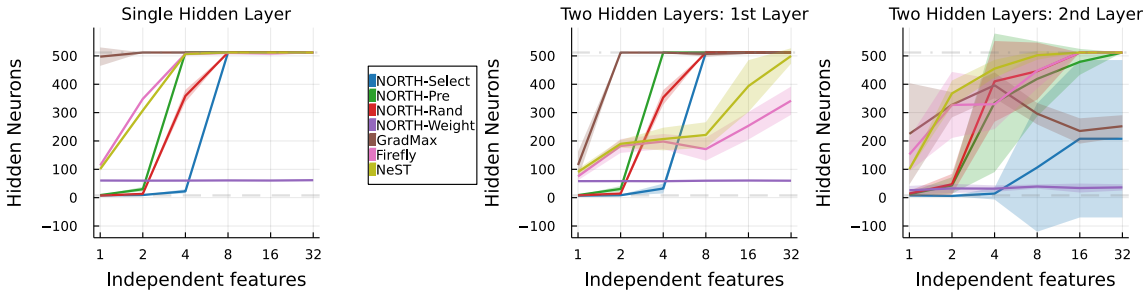


Figure 1: Hidden neurons upon convergence on for MLPs on generated toy data versus the number of independent features in a 1 (left) and 2 (middle, right) hidden layer MLPs.

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 to investigate them in the dynamic context and be able to compare them with our dynamic methods. 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 settings: MLPs (Rumelhart et al., 1986) on generated toy datasets in Section 4.1, MLPs on MNIST (Deng, 2012) in Section 4.2, and convolutional neural networks (CNNs) on image classification: VGG-11 (Simonyan and Zisserman, 2015) and WideResNet-28 (Zagoruyko and Komodakis, 2016) on CIFAR10/CIFAR100 in Section 4.3. 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 Appendix B.1 and a study on hyperparameters in B.2. 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.

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

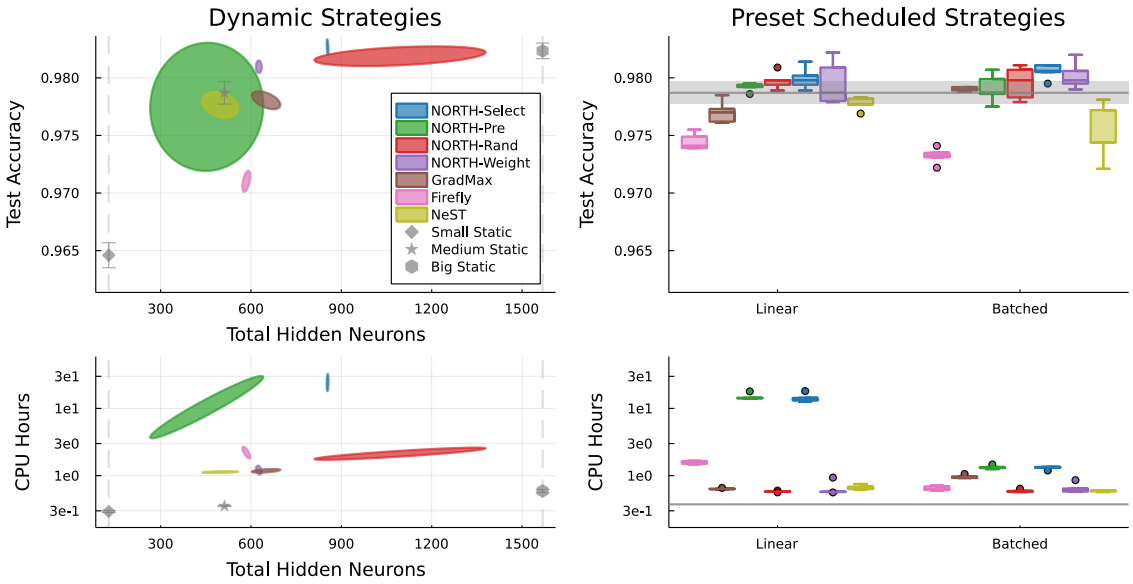


Figure 2: Dynamic (left) and preset scheduled (right) comparisons of network performance across neurogenesis strategies for MLPs on MNIST, evaluated by test accuracy (top) and training time (bottom). Preset scheduled are compared against the Medium Static network, which is the same architecture as the final size of the grown networks.

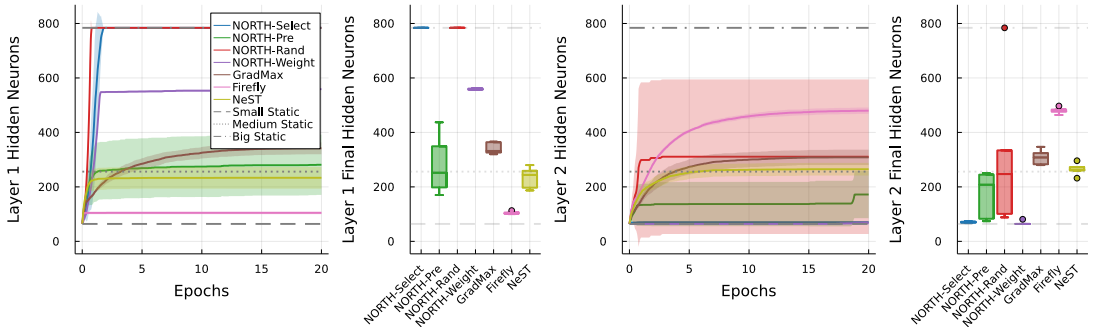


Figure 3: Layer width for MLPs on MNIST over training for dynamic strategies, across training (far-left for Layer 1, center-right for Layer 2) and at the end of training (center-left for Layer 1, far-right for Layer 2).

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.

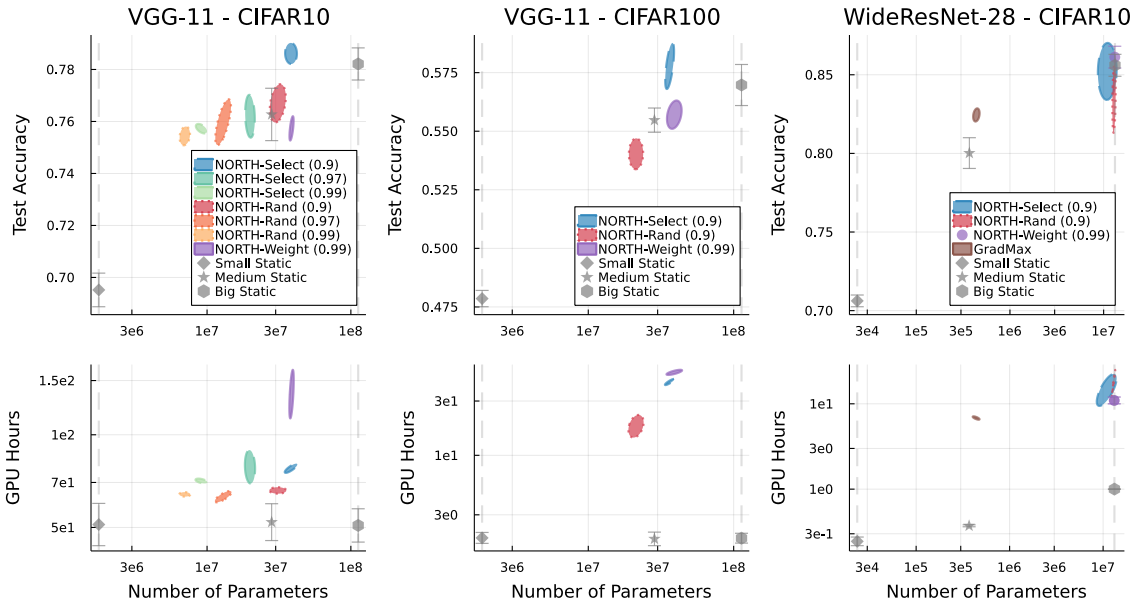


Figure 4: Comparisons of test accuracy (top) and training time (bottom) vs. network size across neurogenesis strategies and values of γ for VGG-11 on CIFAR10 (left) and CIFAR100 (center) and WideResNet-28 on CIFAR10 (right).

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.

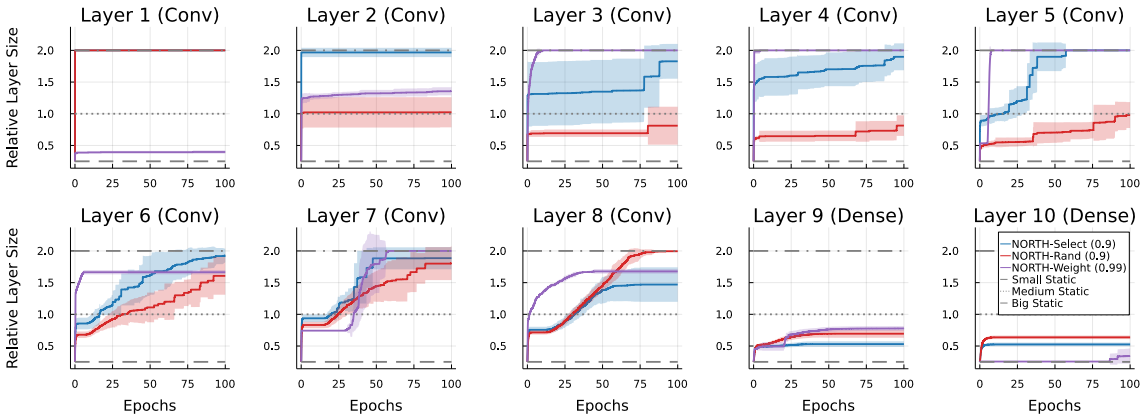
From Figure 3, no dynamic strategy reached the maximum possible network size. Thus, our triggers do respond to the "When" and "How many" questions, converging towards smaller networks. Furthermore, NORTH-Select and NORTH-Random reach networks which are competitive with the largest non-growing network 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 approaches of NORTH-Select and NORTH-Random, as well as NORTH-Weight which has a similar premise towards orthogonal pre-activations.

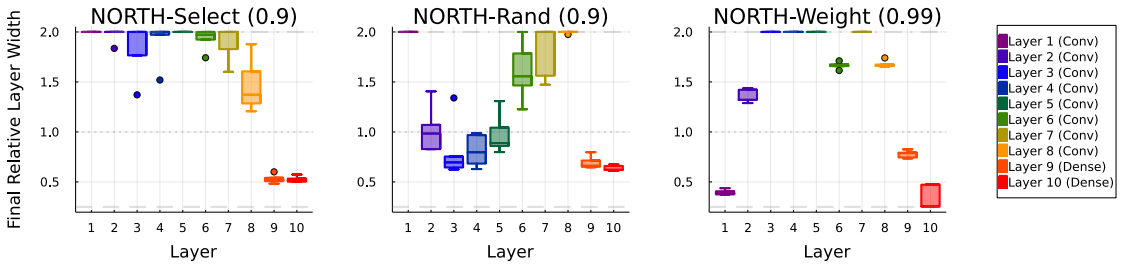
4.3 Deep CNNs on CIFAR10/CIFAR100

We extend neurogenesis to deep convolutional networks and test them on CIFAR10 and CIFAR100 classification with a VGG-11 or WideResNet-28 backbone. 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 and training cost of the different methods is compared against network size for each base architecture and dataset combination in Figure 4. On VGG-11, our NORTH* methods find networks in the same order of size of the standard network, here represented by Medium Static,



(a) Relative layer widths over training.



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

Figure 5: 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).

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 5, 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.

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, 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 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.

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*, volume 18.

Bertoin, D., Bolte, J., Gerchinovitz, S., and Pauwels, E. (2021). Numerical influence of ReLU'(0) on backpropagation. In *Advances in Neural Information Processing Systems*, volume 34, pages 468–479.

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 *4th 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*, volume 34, pages 4896–4906.

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

Deng, L., Li, G., Han, S., Shi, L., and Xie, Y. (2020). Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, 108(4):485–532.

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.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20:1–21.

Evci, U., van Merriënboer, B., Unterthiner, T., Pedregosa, F., and Vladymyrov, M. (2022). GradMax: Growing neural networks using gradient information. In *10th 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.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

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 *10th International Conference on Learning Representations*.

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

Kingma, D. P. and Ba, J. (2015). ADAM: A method for stochastic optimization. In *3rd International Conference on Learning Representations*.

Kumar, A., Agarwal, R., Ghosh, D., and Levine, S. (2021). Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *9th International Conference on Learning Representations*.

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.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. (2017). Pruning filters for efficient convnets. *5th International Conference on Learning Representations*.

Lyle, C., Rowland, M., and Dabney, W. (2022). Understanding and preventing capacity loss in reinforcement learning. In *10th 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 *2nd International Conference on Learning Representations*.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations*.

Wan, A., Dai, X., Zhang, P., He, Z., Tian, Y., Xie, S., Wu, B., Yu, M., Xu, T., Chen, K., et al. (2020). FBNetV2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12965–12974.

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

Wen, W., Yan, F., Chen, Y., and Li, H. (2020). AutoGrow: Automatic layer growing in deep convolutional networks. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 20:833–841.

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*, volume 33, pages 22373–22383.

Wu, L., Wang, D., and Liu, Q. (2019). Splitting steepest descent for growing neural architectures. In *Advances in Neural Information Processing Systems*, volume 32.

Zagoruyko, S. and Komodakis, N. (2016). Wide residual networks. *arXiv preprint arXiv:1605.07146*.

A Supplementary - Methods

A.1 Orthogonality Measure

We additionally study a metric based on the orthogonality gap from Daneshmand et al. (2021) that measures the gap of the covariance matrix of post-activations to the identity matrix. We use an estimate of the orthogonality gap

$$\phi_a^{OG}(f,l) = 1 - \left\| \left(\frac{1}{\|\boldsymbol{H}_l\|_F^2} \right) \boldsymbol{H}_l^\intercal \boldsymbol{H}_l - \left(\frac{1}{\|\boldsymbol{I}_n\|_F^2} \right) \right\|_F, \tag{9}$$

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.

A.2 Activation-based Initializations

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 $\boldsymbol{V}_{Z_l}$ 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 (Bertsekas, 1997) 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.

A.3 Gradient-based Initializations

- NeST from Dai et al. (2019): 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 Wu et al. (2020): 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 Evci et al. (2022): 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.

Table 2: Default hyperparameters.

	MLP - Generated	MLP - MNIST	VGG-11 - CIFAR10	VGG-11 - CIFAR100	WRN-28 - CIFAR10
Optimizer	ADAM (Kingma and Ba, 2015)		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 (Glorot and Bengio, 2010)				
Orthogonality metric	ϕ_{ED}				
ϵ, eq. (2),(4)	0.01				
γ_a, eq. (3)	0.97		0.9, 0.97, 0.99	0.9	
γ_w, eq. (5)	0.99				
Device	CPU		GPU		

A.4 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,

$$T_{act}^{conv}(f, \phi_a, l) = \min\left(0, \left\lfloor M_l \left(\phi_a(f, l) - \gamma_a \max_t \phi_a(f_t, l)\right)\right\rfloor\right). \tag{10}$$

  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 Evci et al. (2022).
- 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.

B Supplementary - Experiments

B.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 2. 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 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 (Bertoin et al., 2021).

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 5 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 (Lacoste et al., 2019; Lannelongue et al., 2021).

B.1.1 Simulated Data. 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.

B.1.2 MLPs on MNIST. The 28×28 images of the MNIST dataset are flattened to 784 features. This is fed into an MLP with 2 hidden layers, then classified into 10 classes representing the digit in the image. We use the standard training and test split.

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 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.

B.1.3 VGG-11 and WideResNet-28 on CIFAR10 and CIFAR100. 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 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 $\frac{1}{4}\times$

Table 3: 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

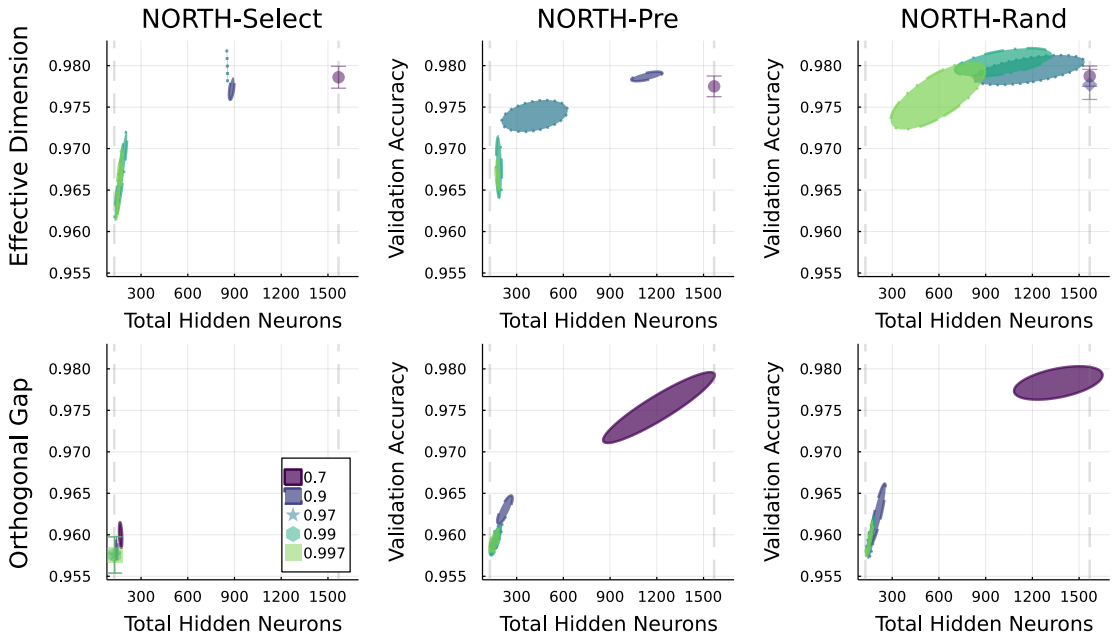
the standard layer widths, also used for the Small Static architecture. The layer sizes for growing networks were limited to 2× the standard layer widths for VGG-11 and 6× the standard layer widths for WideResNet-28-1 (which is WideResNet-28-6 in their notation), also used for the Big Static architecture. 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 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 3. 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.

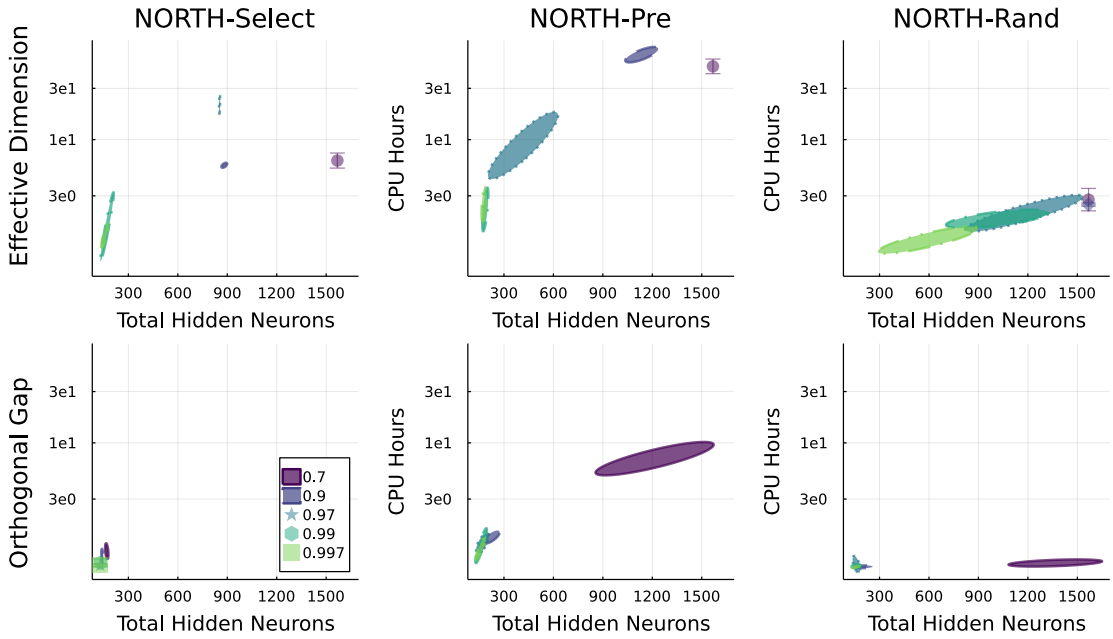
B.2 Hyperparameter studies on MLPs for MNIST

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 4.2. 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 2 versus Orthogonality Gap ϕ_a^{OG} from Equation 9 on performance and training time and their sensitivities to γ_a in Figure 6. 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.



(a) Accuracy versus network size.



(b) Training time versus network size.

Figure 6: Comparison of validation performance and sensitivity to γ_a of ϕ_a^{ED} and ϕ_a^{OG} in activation-based strategies.

We study the effect of batchsize in Figure 7a. 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.

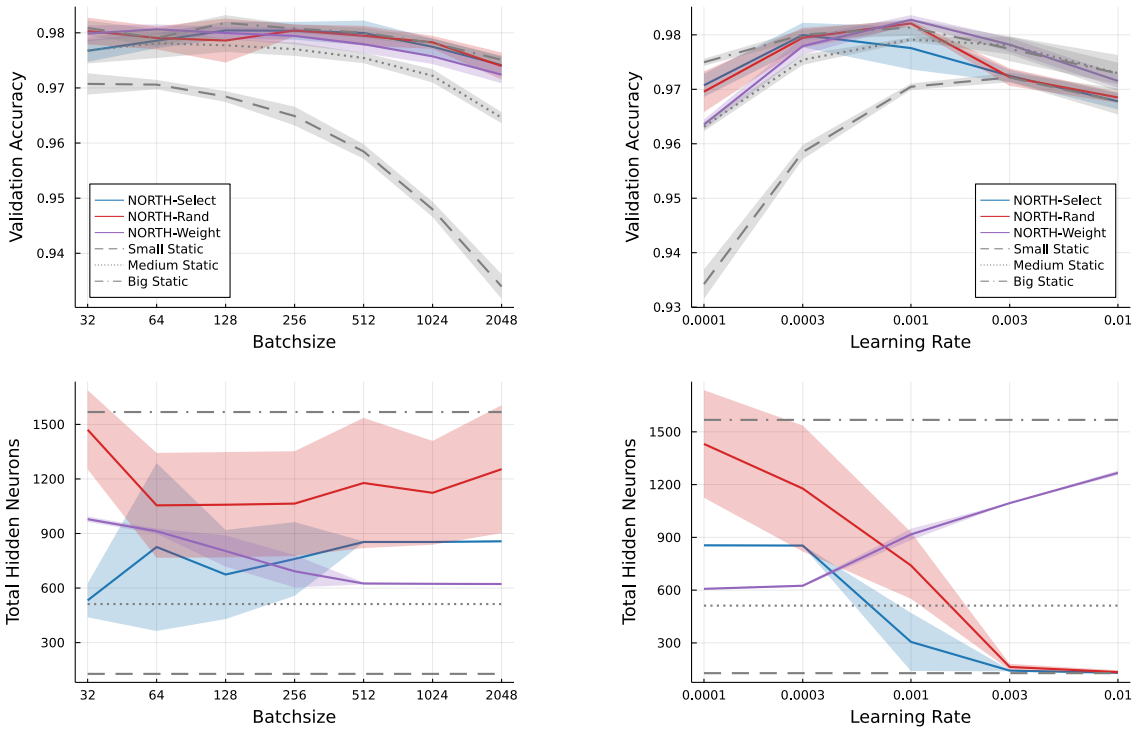


Figure 7: Effects of (a) batchsize and (b) learning rate on NORTH-Select and NORTH-Random, compared to dynamic and static baselines.

The results of various learning rates are shown in Figure 7b. 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.