PROVABLY HIGH DEGREE POLYNOMIAL REPRESENTATION LEARNING WITH FREENETS

Rahul Madhavan
Indian Institute of Science, Bangalore
mrahul@iisc.ac.in

Varun Yerram
Google Research, Bangalore
vyerram@google.com

ABSTRACT

We propose a novel data driven approach to neural architectures based on information flows in a Neural Connectivity Graph (NCG). This technique gives rise to a category of neural networks that we call “Free Networks”, characterized entirely by the edges of an acyclic uni-directional graph. Furthermore, we design a unique, data-informed methodology to systematically prune and augment connections in the proposed architecture during training. We show that any layered feed forward architecture is a subset of the class of Free Networks. Therefore, we propose that our method can produce a class of neural graphs that is a superset of any existing feed-forward networks. Moreover, we analytically examine the expressivity of FreeNets with respect to specific function classes. Our analysis guarantees that FreeNets with k neurons can exactly represent any polynomial of degree k. We perform extensive experiments on this new architecture, to visualize the evolution of the neural topology over real world datasets, and showcase its performance alongside comparable baselines.

1 INTRODUCTION

We posit that conceptualizing a network purely as a layered sequence of neurons is rather constrictive over the search space of all neural architectures. For example, neurons over a single layer may have interconnections. Such a constraint is not necessitated by the functional structure of feed-forward and back-propagation. For these, the only requirement is the existence of an acyclic computation graph, which is the same as the existence of a topological order (not necessarily unique) amongst the neurons in the network (Rigo, 2016). In order to use remove such a constraint over neural connections we propose a novel layer-free graph over neurons — an architecture that we call FREENETS.

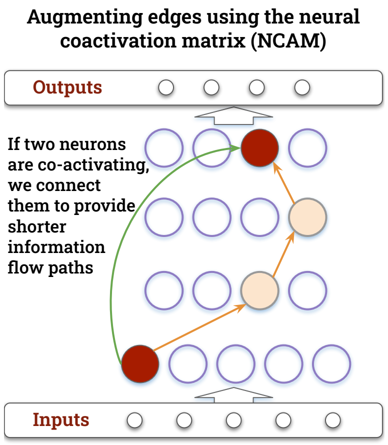
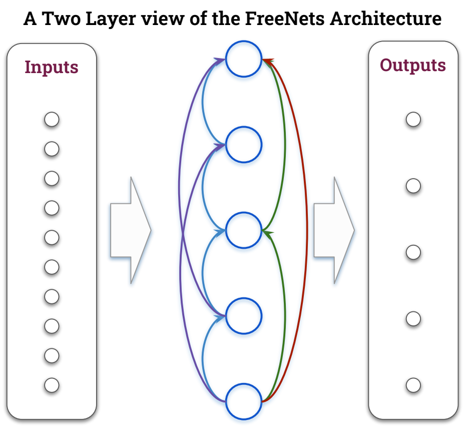

Augmenting edges using the neural coactivation matrix (NCAM)
Outputs
If two neurons are co-activating, we connect them to provide shorter information flow paths
Inputs

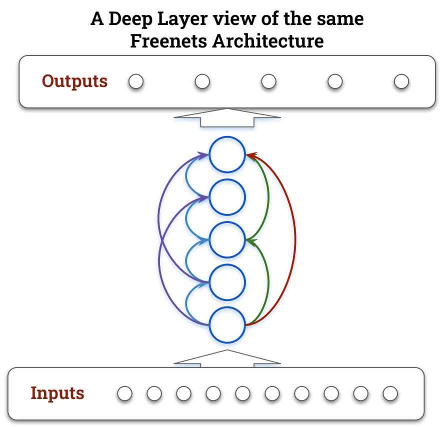
Figure 1: Connecting neurons that capture the same information frees up the path of their co-activation to learn other information.

To initialize the FreeNet architecture, we start with a single parameter: the neuron count within the network. Given such an input, the system yields an initialization characterized by dense interconnections amongst these neurons, adhering to a logical computational sequence. Such a computation sequence is provided by a natural topological ordering amongst the graph nodes. Indeed, the existence of a partial order (a topological order) amongst the nodes in a graph is sufficient to fulfill the acyclicity constraint (Bolte & Pauwels, 2020). This acyclicity, when maintained among the graph nodes, satisfies the essential prerequisites for executing mathematical operations over the network consistently, including back-propagation, which is governed by the chain rule (Dong et al., 2022; Savine, 2019; Schulman et al., 2015). We illustrate such a graph in the Figure 3.

Although this initialization provides a favorable starting point for learning the neural topology, an effective algorithm is requisite to evolve it further. For this we seek an associative learning strategy



(a) Consider a FREENETS initialization with k neurons. Such an initialization may be viewed as a 2 layer neural network which is fully connected to the inputs and outputs



(b) The FREENETS architecture may also be viewed as consisting of k layers, where there are k neuronal compositions before we get to the outputs. This allows for a much richer function space for the outputs.

Figure 2: Different views of the FREENETS architecture, which is the initialization for our proposed Neural Architecture search method.

from neuroscience called Hebbian learning (Wang & Orchard, 2017; Bi & Poo, 2001). Simply put, the Hebb's law states that "Neurons that fire together, wire together".

This suggests firstly that neuronal connections in the brain are not static (unlike in traditional neural networks), and further, that augmenting neuronal connections based on data may be effective. While this provides us with a motivation for adding edges within the neural topology from the neuroscience perspective, one may also view it from the lens of information flow. For two neurons from distinct layers exhibiting a strong activation correlation, information is relayed between them through some (potentially long) path within the neural network. Directly connecting these neurons could maintain their activation patterns, thereby liberating preceding neurons in the pathway to assimilate alternative patterns. While such edge additions resemble residual or skip connections, they have not been systematically explored. Analogously, studies on neuroplasticity in children indicate an initial abundance of neural synapses that diminish over time (Mundkur, 2005). This suggests the presence of numerous non-beneficial neuronal connections that may be transmitting spurious or non-helpful information – underscoring the need for a systematic pruning approach.

The neuron activation patterns, as studied by Maini et al. (2023) suggests an interplay between data, weights and neural architecture. Furthermore, several recent works have studied the dynamics of how training impacts activation of neurons in a network (Stephenson et al., 2021; Baldock et al., 2021; Jiang et al., 2020; Feldman & Zhang, 2020). However, these investigations have predominantly focused on individual neurons rather than information flow. This suggests a potential need to examine the connectivity edges within a neural graph, rather than merely the vertices. To discern the influence of neuronal edges on a network's information flow, we propose an examination of pairwise activation patterns.

For a specified network topology with associated weights, an example might either activate a neuron or leave it dormant. This suggests the representation of an example in the form of a boolean vector within the neural activation domain, denoted as $\boldsymbol{a}_i$ for example i. We can study neural co-activations for this example i through the rank-1 matrix $\boldsymbol{a}_i \cdot \boldsymbol{a}_i^\top$. Analogously, we can construct such matrices for the entire dataset and study the co-activation patterns. Such patterns can tell us where to augment edges between neurons in the network topology, in a manner similar to hebbian learning. Further, by studying $\overline{\boldsymbol{a}_i} \cdot \overline{\boldsymbol{a}_i}^\top$, we can understand when to prune neuronal edges in the connectivity graph.

In summary, we suggest an approach to learn data-dependent neural architectures that encompasses existing neural topologies. We start with simple neurons as building blocks, and iteratively learn a computation graph over these neurons to construct a neural network predictor function. We now lay out the main contributions of this work.

1.1 Our Contributions

The main contributions in the paper are as follows: 1. We propose a novel, learnable Neural network architecture that we call FreeNets. The architecture learning is based on Hebbian learning principle from neuroscience that says neurons that fire together wire together. 2. The FreeNets framework is capable of learning any feed-forward structure conforming to an acyclic computation graph. Hence, we prove that our approach encompasses the capabilities of conventional feedforward architectures. 3. Our network learns the topology and correspondingly, weights for the topology in a manner similar to alternating minimization. 4. The updates that we perform on the neural topology are not necessarily local. Hence, unlike other neural architecture optimization methods that search over neighbours for a suitable update, it is possible for us to move out of local minima through non-local updates. 5. We prove the existence of a class of functions, specifically polynomials of degree k, that can be exactly represented by the FreeNets architecture using k neurons. This result provides insight into the expressive power of FreeNets with respect to certain function classes.

2 Related Work

NAS Methods: Neural Architecture Search (NAS) has emerged as a important framework in the realm of neural architecture design. Several approaches to NAS use RL-based methodologies (You et al., 2020; Zoph et al., 2018; Zhong et al., 2018; Zoph & Le, 2016), including Q-learning (Wu & Jain, 2021), Monte Carlo tree search (Wang et al., 2020), and inverse reinforcement learning (Guo et al., 2019). Liu et al. (2018) introduced differentiable methods for NAS, which has been successfully employed by various others including Wu et al. (2019); Li et al. (2020); Shu et al. (2022); Wang et al. (2021). Some of these methods also consider pruning (Ding et al., 2022).

Such differential methods entail some computational overhead, and several strategies have been proposed to mitigate these, such as parameter sharing (Pham et al., 2018), predicting accuracy without training (Mellor et al., 2021), by analyzing the spectrum of the neural tangent kernel (Chen et al., 2021), accuracy proxies (Abdelfattah et al., 2021), and neural architecture transfer (Lu et al., 2021).

Though gradient-based techniques are prevalent in architecture search, evolutionary network design presents an alternative and historical paradigm in this domain (Stanley & Miikkulainen, 2002). Several recent works have successfully deployed evolutionary strategies including (Real et al., 2017; 2019; Xie & Yuille, 2017). Some noteworthy evolutionary strategies include predictor assisted strategies Peng et al. (2022) covariance based strategies (Sinha & Chen, 2023) and biologically inspired strategies (Amato et al., 2019).

NAS Search Space: A neural architecture can be conceptualized as a macro structure, underpinned by a leitmotif of cells. Many NAS studies, such as Liu et al. (2018); Zoph et al. (2018); Elsken et al. (2019); Cai et al. (2018), have employed a search space centered around cell design. Liu et al. (2017) consider the search space to be the macro-architecture over these cells. Some studies, like (Gao et al., 2019), focus on task-specific architectures such as graph neural networks.

Our work focuses on the macro architecture over neurons, which are the motifs we consider, and is task-agnostic.

Learning Dynamics in neural networks: The learning dynamics of neural networks and their interplay with data has been considered in several recent works on model interpretability, including on the long tail of data (Feldman, 2020; Feldman & Zhang, 2020), on the learnability of examples (Frankle et al., 2020) and on the simplicity bias (Shah et al., 2020) of neural networks. Localization of memory in neural architectures has been studied by (Baldock et al., 2021; Stephenson et al., 2021). On the other hand, Maini et al. (2023) propose than memorization of examples by neural networks can not be localized to a few layers, but is often determined by a small set of neurons that may be distributed across layers. Sinitsin et al. (2020) consider modifications to the architecture in order to improve long tail efficiency.

3 Methodology

In this section, we describe the method through which FreeNets is initialized and evolves.

The FreeNets architecture is a dense computation graph over the input. We ensure no cycles by requiring that the initial adjacency matrix for this graph be lower triangular. Note that we do not allow for cycles or self-loops, and hence the diagonal entries of such an adjacency matrix are 0. In fact, we will show that the FreeNets architecture is the most complex function one can learn with k neurons (or k computational nodes). This is a good initialization from which to learn about inter-neuronal interactions as every neuron is free to interact with every other neuron (through the weights) for all points in the dataset.

3.1 Motivation for the FreeNets architecture

Traditional architectures have prematurely optimized for the number of weights in the network by handcrafting layer-design. Popular architectures use decreasing layer widths as we move from the inputs to the outputs. Other architectures use an encoder-decoder model which encodes the inputs into a smaller representation space, before decoding these into an output. But we argue that such connections are pre-mature before looking at the data. Indeed, synaptic plasticity is highest at birth and decreases with time Cutler & Mattson (2006). Yet, cognitive function goes up with time, as the same neurons evolve to a sparser connectivity. This neuroscience motivation indicates that beyond requiring a robust initialization, there is also a need for an effective data-dependent learning method to evolve the neuronal topology.

We study this question in the next three sub-sections.

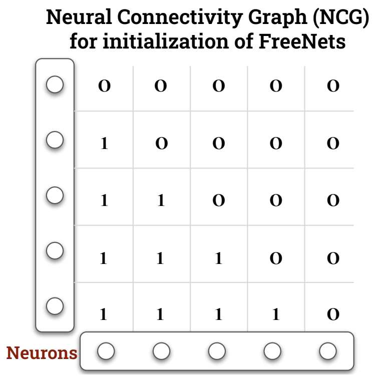

Neural Connectivity Graph (NCG) for initialization of FreeNets
0 0 0 0 0
1 0 0 0 0
1 1 0 0 0
1 1 1 0 0
1 1 1 1 0
Neurons

Figure 3: The adjacency matrix for FreeNets initialization is lower triangular, dense and cycle-free.

3.2 "Neurons that activate together, should be wired together"

While Hebbian learning emphasizes enhancing neuronal connection weights based on synchronous activations, contemporary neural models predominantly rely on backpropagation for weight adjustment. We suggest that both the perspectives are pertinent, but pertain to distinct variables. A neural function encompasses two discrete variable sets: the edges spanning the neurons and the associated weights for the selected edges. Back propagation helps to optimize the set of weights associated with a chosen set of edges in a neural topology. On the other hand, Hebbian learning may be used to adjust this edge set. Indeed, one may seek to optimize over these two distinct objectives, in a manner similar to alternating minimization.

We introduce here the notation of a neural connectivity graph NCG, which specifies the edge set between k neurons in the network as a binary adjacency matrix. Note that $NCG \in \{0,1\}^{k\times k}$.

We can also look at augmenting interneuronal connections from an information flow perspective. Consider two neurons in far away layers that are disconnected, but are firing together for certain examples: see Figure 1. There exists some path where information is flowing to both of these neurons (for the said examples). Such a path may be long, and may employ many other neurons. By directly connecting the neuron that is computed first to the later neuron, we may shorten the information flow path. Further, we may use the earlier path to capture other information, thereby expanding the information capacity of the network.

3.3 Pruning Neuronal Connections

While we have looked at interneuronal edge augmentation through hebbian learning, we might ask: "is there any way to prune edges in the NCG"? Consider neurons that are mostly inactive together. Such neurons may co-activate for a small set of examples, and then the information may be passed deeper into the network. But for this small subset of examples, one may argue that since they have already been identified (by the first neuron), this information can be directly passed to the deeper layers, or to the outputs. Such a method suggests that depth in a network is valuable, and hence it should be used for examples occurring more commonly. Sparse examples may simply be memorised by early neurons (and passed to outputs). Hence we suggest that neurons that are jointly inactive but share an edge between them, should have the edge pruned.

Now we look at how to construct statistics regarding such co-activation in the next subsection.

Neuron 1 Activation	**Neuron 2** Activation	$1-2$ **Connectivity**	**Update to Perform**	**Explanation**
0	0	0	Do not augment	They are not connected and not firing together, so nothing to do
0	0	1	Prune $1-2$ edge	Both are inactive neurons. In the backward pass, the weight cannot be updated. Hence prune.
0	1	0	Do not augment	Need not connect. Neuron 2 is being activated by other neurons.
0	1	1	Do not prune	Weights can be updated on back-prop.
1	0	0	Do not augment	Neuron 1 may be activating other neurons. Further, an added edge-weight may not be updated during back-propagation.
1	0	1	Do not prune	They are connected, and not firing together. Yet the first neuron may be inactivating the second neuron.
1	1	0	Add $1-2$ edge	We can add an edge to shorten the information flow path.
1	1	1	Do not prune	They are connected and firing together, so nothing to do.

Table 1: Summary of NCG updates based on NCAM and NCIM. Assume that neuron 1 comes before neuron 2 in the forward computation graph.

3.4 Construction of the Neural Co-activation and co-inactivation matrices

We introduce some notation that will aid in understanding the construction of the neural co-activation matrix NCAM, and neural co-inactivation matrix NCIM. Let the data $\mathcal{D}$ to the neural network come in the form of samples from a space $\mathcal{X} \times \mathcal{Y}$, such that $\mathcal{D} := \{(x_1, y_1), \ldots, (x_n, y_n)\}$ where each $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. Let us consider a neural graph $\mathcal{G}$ consisting of k neurons and ℓ edges, with number of weights ℓ. Let $\texttt{NN}_{\mathcal{G}} : \mathcal{X} \to \mathcal{Y}$ be the estimator of the function mapping $\mathcal{X}$ and $\mathcal{Y}$. $\texttt{NN}_G$ is parameterized with ℓ weights $\{w_1, \ldots, w_\ell\}$ where $w_i \in \mathbb{R}$ for all $i \in [\ell]$.

Let $w^* = \arg\min_{w \in \mathbb{R}^\ell} \|y - \texttt{NN}_{\mathcal{G}}(x)\|_2^2$ be the optimal set of weights. Now we can define $\texttt{NN}^*_{\mathcal{G}}$ as the neural function parameterized by an optimal set of weights. Call $\texttt{NA}^* : \mathcal{X} \to \{0,1\}^k$ the neural activation function, where $\texttt{NA}^*$ is a function of $\texttt{NN}_{\mathcal{G}}$ and w^*. For ease of notation, we assume we are always working with w^* and write NA to be the neural activation function.

Now we do a forward pass over the data and record activations of the k neurons. Let $\mathcal{S} \in \{0,1\}^{k \times n} := \{\texttt{NA}^*(x_i) \text{ for all } x_i \in \mathcal{D}\}$. Now we are ready to define the NCAM matrix as $\texttt{NCAM} := 1/n \cdot \mathcal{S} \cdot \mathcal{S}^\top$. Note that the computation of NCAM can be parallelized over the examples as follows. We note: $\texttt{NCAM} = 1/n \cdot \sum_{i \in [n]} \texttt{NA}(x_i) \cdot \texttt{NA}(x_i)^\top$. Analogously, if $\overline{\mathcal{S}} := \neg\mathcal{S}$, then $\texttt{NCIM} := 1/n \cdot \overline{\mathcal{S}} \cdot \overline{\mathcal{S}}^\top$. Therefore, $\texttt{NCIM} = 1/n \cdot \sum_{i \in [n]} \overline{\texttt{NA}(x_i)} \cdot \overline{\texttt{NA}(x_i)}^\top$ where $\overline{\texttt{NA}(x_i)} := \neg\texttt{NA}(x_i)$.

Now that we have recorded the neural co-activations, we are in a position to consider what updates to make to the neural connectivity graph NCG based on these. We can threshold the NCAM and NCIM values at a certain preset values. For each (i, j) pair of neurons we define them to be co-activating if $\texttt{NCAM}(i, j) > 1 - \varepsilon$. Similarly, they are said to be jointly inactive if $\texttt{NCIM}(i, j) > 1 - \varepsilon$. Based on these values, we propose an update table to the neural connectivity graph in Table 1. In a matrix form, these may be written as $\mathbf{Aug} := (\texttt{NCAM} > 1 - \varepsilon) \wedge \neg\mathcal{G}$ and $\mathbf{Prune} := (\texttt{NCIM} > 1 - \varepsilon) \wedge \mathcal{G}$, where $\mathbf{Aug}, \mathbf{Prune}$ are the augmenting and pruning indicator matrices respectively. This defines our update equation to be $\mathcal{G}' = (\mathcal{G} \wedge \neg\mathbf{Prune}) \vee (\mathbf{Aug} \wedge \neg\mathbf{Prune})$. We expand this reasoning into the Algorithms below.

4 Algorithm and Theory

We now propose the algorithms pertaining to FreeNets. Algorithm 1, takes as input a dataset $\mathcal{D}$ and a certain number of neurons k. It calls Algorithm 2 for updates to the neural connectivity graph.

4.1 Main Theorem

We now state a theorem regarding the expressive power of FreeNets.

Theorem 1. *Let $D \subseteq \mathbb{R}$ be a compact domain, and let $\mathcal{P}_k$ denote the set of all univariate polynomials of degree at most k over D. Then the* FreeNets *architecture with k neurons and a squared ReLU activation function $\sigma(x) = (\max\{0, x\})^2$ can exactly represent any polynomial $p \in \mathcal{P}_k$ over D.*

Algorithm 1 Find best FREENETS architecture using data

```
1: Input:
   (1) Data $\mathcal{D} := \{(x_1, y_1), \ldots, (x_n, y_n)\}$ where each $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$.
   (2) Number of neurons: $k$.
2: Initialize $\mathcal{G}$ := FREENETS($k$) for $k$ neurons
3: while $\mathcal{G}$ does not converge do
4:     $\mathcal{G}$ := FREEEVOLVE($\mathcal{D}, \mathcal{G}$)
5: end while
6: return $\mathcal{G}$
```

Proof. Our goal is to show that any polynomial $p(x)$ of degree at most k can be exactly represented by an FREENETS network with k neurons using the squared ReLU activation function.

CONSTRUCTION: Consider the polynomial $p(x) = \sum_{d=0}^{k} a_d x^d$.

We will construct neurons that compute monomials x^d for $d = 1, 2, \ldots, k$ and then combine them linearly to form $p(x)$.

GENERATING MONOMIALS: We assume $x \geq 0$ for all $x \in D$ (if not, we can shift x appropriately using the biases). The squared ReLU activation function is defined as $\sigma(x) = x^2$ for $x \geq 0$ and zero otherwise. Let $h_0 = x$. We will construct neurons $h_1, h_2, \ldots, h_k$ such that $h_d = x^{2^d}$.

Base Case: For $d = 1$, $h_1 = \sigma(h_0) = \sigma(x) = x^2$.

Inductive Step: Assume $h_{d-1} = x^{2^{d-1}}$. Then,

$$h_d = \sigma(h_{d-1}) = \sigma(x^{2^{d-1}}) = \left(x^{2^{d-1}}\right)^2 = x^{2^d}.$$

Thus, we can generate monomials whose degrees are powers of 2 up to 2^k.

GENERATING INTERMEDIATE DEGREES: To obtain monomials of degrees that are not powers of 2, we can utilize linear combinations of previously computed monomials. Since we can add outputs of neurons (due to the fully connected nature of FREENETS), we can exploit the binomial expansion:

$$\sigma(ah_i + bh_j) = (ah_i + bh_j)^2 = a^2 h_i^2 + 2ab h_i h_j + b^2 h_j^2.$$

By appropriately choosing coefficients a and b, we can produce cross terms $h_i h_j$, which correspond to monomials of degrees $\deg(h_i) + \deg(h_j)$.

CONSTRUCTING THE POLYNOMIAL: Once all monomials up to degree k are generated, the output layer combines them linearly to form $p(x)$:

$$y = \sum_{d=1}^{k} v_d h_d + c,$$

where v_d are the coefficients adjusted to match a_d in $p(x)$, and $c = a_0$ is the constant term.

CONCLUSION: Therefore, the FREENETS network with k neurons can represent any polynomial $p(x)$ of degree at most k exactly over D. $\square$

We now make a simple claim regarding pruning and augmentation in the network.

Claim 4.1. Consider any consistent[1] neural topology over n neurons. Let $\mathcal{G}$ be the neural graph corresponding to this architecture. Further, let $\mathcal{H}_n$ to be class of neural topologies over n neurons reachable by Algorithm 1 for some dataset. Then $\mathcal{G} \in \mathcal{H}_n$.

Algorithm 2 FREEEVOLVE: Update Algorithm for FREENETS using Neural Coactivations

```
1: Input:
   (1) Data $\mathcal{D} := \{(x_1, y_1), \ldots, (x_n, y_n)\}$ where each $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$.
   (2) A neural topology $\mathcal{G}$ consisting of $k$ neurons and $\ell$ edges.
2: Train the weights of edges in $\mathcal{G}$ until convergence.
   i.e. Let $w^* = \arg\min_{w \in \mathbb{R}^\ell} \|y - \texttt{NN}_\mathcal{G}(x)\|_2^2$ where $\texttt{NN}_\mathcal{G} : \mathcal{X} \to \mathcal{Y}$ is the neural network function over $\mathcal{G}$.
   Call $\texttt{NA} : \mathcal{X} \to \{0,1\}^k$ the Neural activation function, where $\texttt{NA}$ is a function of $\texttt{NN}_\mathcal{G}$ and $w^*$.
3: Do a forward pass over the data and record activations of the k neurons.
   Let $\mathcal{S} \in \{0,1\}^{k \times n} := \{\texttt{NA}(x_i)$ for all $x_i \in \mathcal{D}\}$
4: Compute $\texttt{NCAM} := \frac{1}{n}\mathcal{S} \cdot \mathcal{S}^\top$ and $\texttt{NCIM} := \frac{1}{n}\overline{\mathcal{S}} \cdot \overline{\mathcal{S}}^\top$ where $\overline{\mathcal{S}} := \neg\mathcal{S}$.
   These are easily computed as $\texttt{NCAM} = \frac{1}{n}\sum_{i \in [n]} \texttt{NA}(x_i) \cdot \texttt{NA}(x_i)^\top$ and
   $\texttt{NCIM} = \frac{1}{n}\sum_{i \in [n]} \overline{\texttt{NA}(x_i)} \cdot \overline{\texttt{NA}(x_i)}^\top$ where $\overline{\texttt{NA}(x_i)} := \neg\texttt{NA}(x_i)$
5: Compute the augmenting and pruning matrices.
   $\mathbf{Aug} := (\texttt{NCAM} > 1 - \varepsilon) \wedge \neg\mathcal{G}$
   $\mathbf{Prune} := (\texttt{NCIM} > 1 - \varepsilon) \wedge \mathcal{G}$
6: Update the graph $\mathcal{G}$.
   $\mathcal{G}' = (\mathcal{G} \wedge \neg\mathbf{Prune}) \vee (\mathbf{Aug} \wedge \neg\mathbf{Prune})$
7: return $\mathcal{G}'$
```

The proof of this claim lies in the fact that every neural architecture is a subset of the edges in FREENETS: See Figure 3. This shows that any layered feed-forward architecture is reachable by Algorithm 1 starting from FREENETS. This leads us to the question of what we can say about the fixed points of the algorithm. We now show some properties of this fixed point.

Theorem 2. Let $\mathcal{G}^* = \langle\{1, \ldots, n\}, \mathbf{E}\rangle$ be the output of Algorithm 1 where $\mathbf{E}$ is the set of edges in $\mathcal{G}^*$. Then $\mathbf{Aug} \wedge \neg\mathbf{Prune} \subseteq \mathbf{E}$

Proof Sketch. The proof of this claim lies in the characterization of the fixed point. If there are any edges in $\mathbf{Aug} \wedge \neg\mathbf{Prune}$, then such edges are added to $\mathcal{G}^*$, and hence the graph is updated by Algorithm 2. Therefore at convergence, $\mathbf{Aug} \wedge \neg\mathbf{Prune}$ is a sub-graph of $\mathcal{G}^*$. □

5 EXPERIMENTS

Datasets and Models: We perform experiments on 3 Image Classification datasets, MNIST (Deng, 2012), FashionMNIST (Xiao et al., 2017) and a truncated version of Extended-MNIST (Cohen et al., 2017). We convert the images into pixel vectors to train the proposed architecture as well as the baselines. To evaluate FREENETS, we compare them with fully connected neural networks (FCNNs). For a model with k neurons, we map the data to k dimensions using a fully connected layer. In case of FREENETS, we have interneuronal connections as given by the neural connectivity graph. Finally we convert these k activations, to the class probabilities using a fully connected layer. To ensure fair comparison between FREENETS and FCNN, we add extra neurons to FCNN if the number of parameters defer substantially.

Model and Training setup: The FREENETS architectures consists of 3 essential components: an encoder – which is a linear map, the FREENET connections, and a decoder – also a linear map. The encoder maps the input to the FREENET and the decoder maps the FREENET activations into the output logits. The weights and the neural connectivity graph help us compute the activation values of the FREENET for any given input. We use these activation values to calculate $\mathcal{S}$ in the FREEEVOLVE algorithm and modify the architecture. The activations are then decoded, again using a linear map, to generate class probabilities. To allow efficient modification of architecture, we store the weights of all connections during training, and mask the weights that are pruned at a particular step. During inference, we propose to store only the weights required for the computation, thereby reducing the FLOPS for computation.

All weight matrices were initialized with a Xavier Initializer (Glorot & Bengio, 2010) while the biases were drawn from a uniform distribution with fixed support.

[1]a consistent neural topology which is causal and allows backpropagation, has an acyclic computation graph

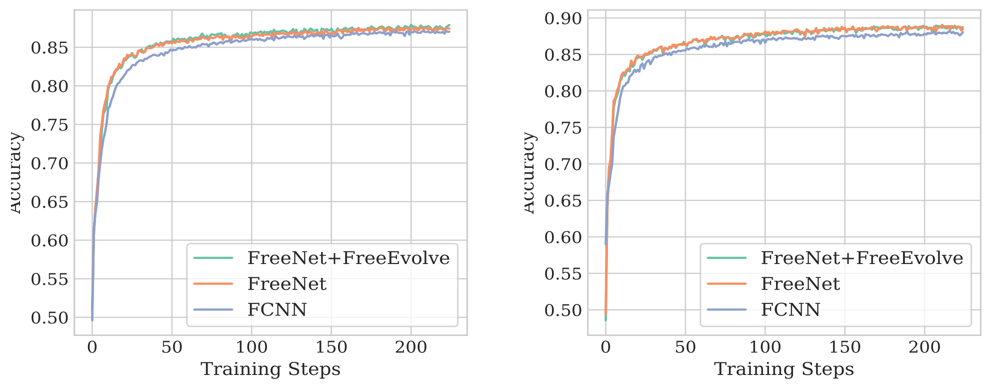


Figure 4: Variation of Validation accuracy on FashionMNIST for architectures with (i) 40 Neurons (ii) 90 Neurons. The Training curve of FREEEVOLVE approximately stays above the FCNN curve.

Model Training In the Figure 4, we compare the proposed architecture with FCNNs in terms of training convergence. We generally find the best FREENETS outperform the best FCNN for the same number of neurons, thereby suggesting the superior connectivity of neurons in our architecture.

Variation of Accuracy and number of neurons Figure 5 plots the accuracy with number of neurons in the network for two tasks, viz. FashionMNIST and MNIST. We observe that for FashionMNIST dataset, the performance gap between FREENETS and FCNN increases by adding more neurons to the model. We observe that for neural networks with sufficient representational capactiy, the MNIST dataset is straightforward to process. Hence, different approaches converge to similaries accuracies on this task. Consequently, we can observe our model improvements more distinctly on the harder datasets – viz. FashionMNIST and EMNIST.

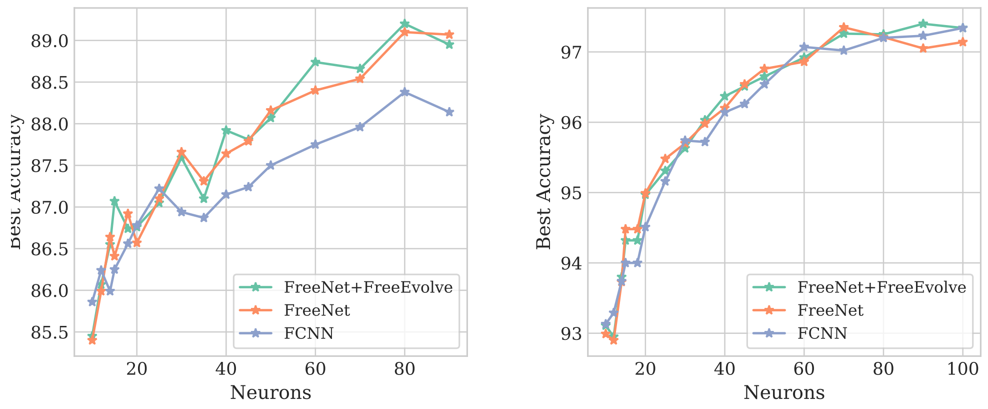


Figure 5: Variation of Accuracy with Number of neurons for (i) FashionMNIST (ii) MNIST

Evolution of Neural co-activations during Training: To examine the evolution of different components of the algorithm, we study the variation of number of weights pruned and augmented with the number of training steps. The plot in Figure 6, clearly suggests the convergence of the FREEEVOLVE algorithm. We observe that at the start of the training, a significant portion of model weights are pruned. To offset that effect, the augmenting matrix acts after the pruning step to correct the model. Both these matrices, complement each other, in order to find the optimal architecture for FREENETS. Near the end of training – when the model has converged – the pruning and augmenting matrices stop influencing the NCG and hence no more weights are added or removed. At this point, the model can be optimized (for optimal weights) through gradient based algorithms.

Dataset	Model	30	35	40	45	50	60	70	80	90	100
MNIST	FreeEvolve	95.63	96.03	96.37	96.51	96.65	96.92	97.26	97.25	97.4	97.34
	FCNN	95.74	95.72	96.14	96.26	96.54	97.07	97.02	97.2	97.23	97.34
FMNIST	FreeEvolve	87.59	87.1	87.92	87.81	88.07	88.74	88.66	89.2	88.95	88.89
	FCNN	86.94	86.87	87.15	87.24	87.5	87.75	87.96	88.38	88.14	88.23
EMNIST	FreeEvolve	78.77	80.75	81.52	81.38	82.58	83.33	84.27	84.23	85.29	85.51
	FCNN	78.55	79.97	80.63	81.92	81.7	82.63	84.07	84.12	84.6	84.98

Table 2: In the above table, we detail the accuracy of various models with the number of neurons in the model, segmented across the three datasets.

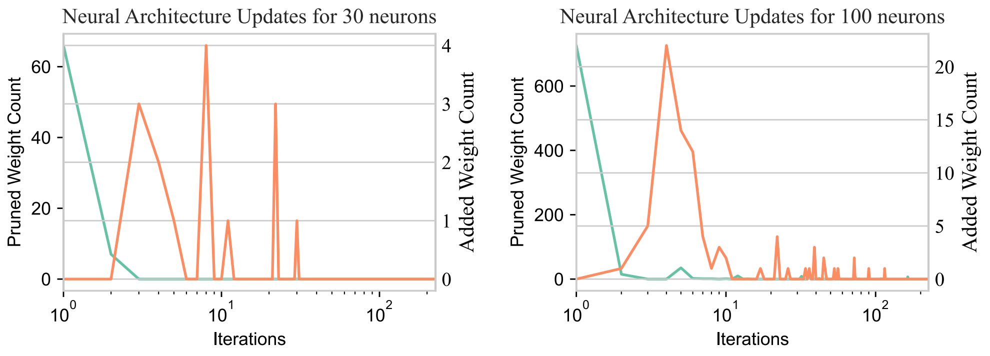


Figure 6: Per-iteration Prune(Green)/Add(Orange) weight count for the model with 30 Neurons (left) and 100 neurons (right).

Prune Matrix and Augment Matrix As the algorithm progresses, we expect the values of the pruning and addition matrices to evolve to higher values. To experimentally verify this hypothesis, we run our training at $\varepsilon = 0.25$ and find the kernel density estimates (KDE) for the NCG matrix, and how they evolve. Our plot in Figure 7 suggests a significant shift in kernel density to higher values in the matrices through the neural connectivity evolution as given in Algorithm 1.

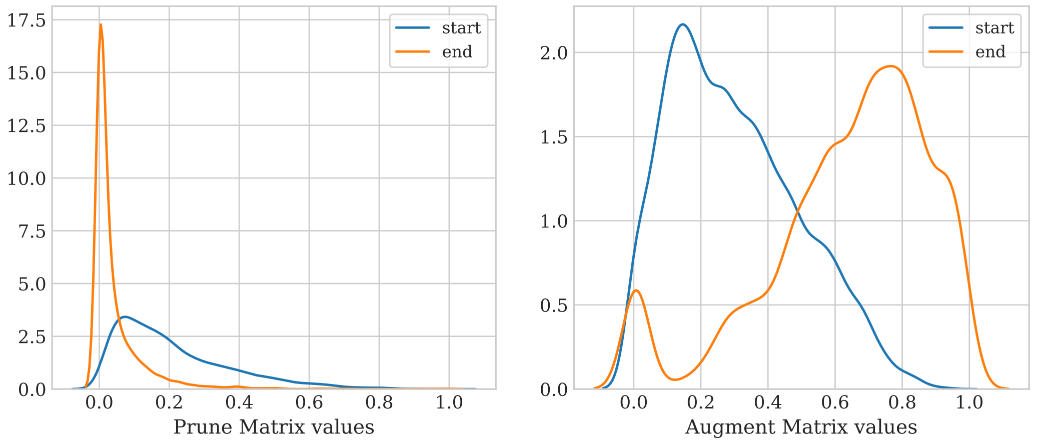


Figure 7: KDE Plots of Prune and Augment matrices at the start and end of the training schedule. The values of the Prune matrix reduce, which implies that the constraints posed by the FreeEvolve algorithm are more easily satisfied as the training progresses. The same can be concluded about the Augment matrix

Through this experiment, we are able to conclude that the model is able to adapt to the pruning and augmenting of connections, thereby producing the activations to satisfy our co-activation threshold, ε. The prune matrix, consists of pairs of neurons that are jointly inactive, and the augment matrix consists of pairs that are co-activating. By observing that the average density in the prune matrix is shifting (sharply) towards the left, and the density in the add matrix is moving towards higher values, we suggest that the evolution of FreeNets successfully enhances the information flow in the network.

6 Conclusion and Future Work

In summary, we have proposed a neural architecture search method that we initialize through a graph over individual neurons. The neural architecture evolves through the method of Neural Co-Activation and Neural Co-inactivation (`NCAM` and `NCIM` respectively), which we use to augment and prune neuronal edges. Our method can achieve non-local updates to the neural architecture, which may help it to evade local minima. We contend that our framework embodies a macro neural architecture search approach, which may be adaptable to include convolutional cells or RNN blocks based on task requirements. Thus, we assert that the FreeNets framework offers the versatility to accommodate a diverse range of neural functionalities in a neuron efficient manner.

References

Mohamed S Abdelfattah, Abhinav Mehrotra, Lukasz Dudziak, and Nicholas D Lane. Zero-cost proxies for lightweight nas. *arXiv preprint arXiv:2101.08134*, 2021.

Giuseppe Amato, Fabio Carrara, Fabrizio Falchi, Claudio Gennaro, and Gabriele Lagani. Hebbian learning meets deep convolutional neural networks. In *Image Analysis and Processing–ICIAP 2019: 20th International Conference, Trento, Italy, September 9–13, 2019, Proceedings, Part I 20*, pp. 324–334. Springer, 2019.

Robert Baldock, Hartmut Maennel, and Behnam Neyshabur. Deep learning through the lens of example difficulty. *Advances in Neural Information Processing Systems*, 34:10876–10889, 2021.

Guo-qiang Bi and Mu-ming Poo. Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual review of neuroscience*, 24(1):139–166, 2001.

Jérôme Bolte and Edouard Pauwels. A mathematical model for automatic differentiation in machine learning. *Advances in Neural Information Processing Systems*, 33:10809–10819, 2020.

Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-level network transformation for efficient architecture search. In *International Conference on Machine Learning*, pp. 678–687. PMLR, 2018.

Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. *arXiv preprint arXiv:2102.11535*, 2021.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pp. 2921–2926. IEEE, 2017.

Roy G Cutler and Mark P Mattson. Introduction: The adversities of aging. *Ageing research reviews*, 5(3):221–238, 2006.

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

Yadong Ding, Yu Wu, Chengyue Huang, Siliang Tang, Fei Wu, Yi Yang, Wenwu Zhu, and Yueting Zhuang. Nap: Neural architecture search with pruning. *Neurocomputing*, 477:85–95, 2022.

Zehao Dong, Muhan Zhang, Fuhai Li, and Yixin Chen. Pace: A parallelizable computation encoder for directed acyclic graphs. In *International Conference on Machine Learning*, pp. 5360–5377. PMLR, 2022.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

Vitaly Feldman. Does learning require memorization? a short tale about a long tail. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pp. 954–959, 2020.

Vitaly Feldman and Chiyuan Zhang. What neural networks memorize and why: Discovering the long tail via influence estimation. *Advances in Neural Information Processing Systems*, 33:2881–2891, 2020.

Jonathan Frankle, David J Schwab, and Ari S Morcos. The early phase of neural network training. *arXiv preprint arXiv:2002.10365*, 2020.

Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graphnas: Graph neural architecture search with reinforcement learning. *arXiv preprint arXiv:1904.09981*, 2019.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Minghao Guo, Zhao Zhong, Wei Wu, Dahua Lin, and Junjie Yan. Irlas: Inverse reinforcement learning for architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9021–9029, 2019.

Ziheng Jiang, Chiyuan Zhang, Kunal Talwar, and Michael C Mozer. Characterizing structural regularities of labeled data in overparameterized models. *arXiv preprint arXiv:2002.03206*, 2020.

Liam Li, Mikhail Khodak, Maria-Florina Balcan, and Ameet Talwalkar. Geometry-aware gradient algorithms for neural architecture search. *arXiv preprint arXiv:2004.07802*, 2020.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

Zhichao Lu, Gautam Sreekumar, Erik Goodman, Wolfgang Banzhaf, Kalyanmoy Deb, and Vishnu Naresh Boddeti. Neural architecture transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):2971–2989, 2021.

Pratyush Maini, Michael C Mozer, Hanie Sedghi, Zachary C Lipton, J Zico Kolter, and Chiyuan Zhang. Can neural network memorization be localized? *arXiv preprint arXiv:2307.09542*, 2023.

Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In *International Conference on Machine Learning*, pp. 7588–7598. PMLR, 2021.

Nandini Mundkur. Neuroplasticity in children. *The Indian Journal of Pediatrics*, 72:855–857, 2005.

Yameng Peng, Andy Song, Vic Ciesielski, Haytham M Fayek, and Xiaojun Chang. Pre-nas: Evolutionary neural architecture search with predictor. *IEEE Transactions on Evolutionary Computation*, 27(1):26–36, 2022.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pp. 4095–4104. PMLR, 2018.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International conference on machine learning*, pp. 2902–2911. PMLR, 2017.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Michel Rigo. *Advanced graph theory and combinatorics*. John Wiley & Sons, 2016.

Antoine Savine. Computation graphs for aad and machine learning part i: Introduction to computation graphs and automatic differentiation. *Wilmott*, 2019(104):36–61, 2019.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. *Advances in neural information processing systems*, 28, 2015.

Harshay Shah, Kaustav Tamuly, Aditi Raghunathan, Prateek Jain, and Praneeth Netrapalli. The pitfalls of simplicity bias in neural networks. *Advances in Neural Information Processing Systems*, 33:9573–9585, 2020.

Yao Shu, Zhongxiang Dai, Zhaoxuan Wu, and Bryan Kian Hsiang Low. Unifying and boosting gradient-based training-free neural architecture search. *Advances in Neural Information Processing Systems*, 35:33001–33015, 2022.

Nilotpal Sinha and Kuan-Wen Chen. Neural architecture search using covariance matrix adaptation evolution strategy. *Evolutionary Computation*, pp. 1–25, 2023.

Anton Sinitsin, Vsevolod Plokhotnyuk, Dmitriy Pyrkin, Sergei Popov, and Artem Babenko. Editable neural networks. *arXiv preprint arXiv:2004.00345*, 2020.

Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Cory Stephenson, Suchismita Padhy, Abhinav Ganesh, Yue Hui, Hanlin Tang, and SueYeon Chung. On the geometry of generalization and memorization in deep neural networks. *arXiv preprint arXiv:2105.14602*, 2021.

Lin Wang and Jeff Orchard. Investigating the evolution of a neuroplasticity network for learning. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(10):2131–2143, 2017.

Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Neural architecture search using deep neural networks and monte carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 9983–9991, 2020.

Ruochen Wang, Minhao Cheng, Xiangning Chen, Xiaocheng Tang, and Cho-Jui Hsieh. Rethinking architecture selection in differentiable nas. *arXiv preprint arXiv:2108.04392*, 2021.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10734–10742, 2019.

Robert Wu and Rohan Jain. Qenas: Q-learning for efficient neural architecture search. Technical report, Technical report, Department of Computer Science, University of Toronto, 2021.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

Lingxi Xie and Alan Yuille. Genetic cnn. In *Proceedings of the IEEE international conference on computer vision*, pp. 1379–1388, 2017.

Shan You, Tao Huang, Mingmin Yang, Fei Wang, Chen Qian, and Changshui Zhang. Greedynas: Towards fast one-shot nas with greedy supernet. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1999–2008, 2020.

Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2423–2432, 2018.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

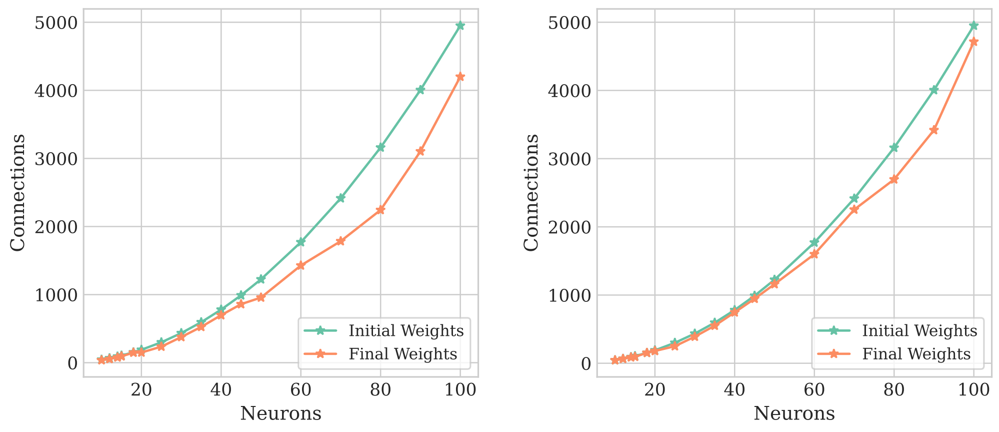


Figure 8: Number of connections i.e. weights in the network at the start and end of training steps for various Neuron sizes (i) FashionMNIST (ii) MNIST

A Comparison of Initial and Final number of Weights with number of neurons

In this section, we analyze the evolution of neural connections during training by comparing the number of connections between neurons at the beginning and end of the training process for varying neuron sizes. The plots in Figure 8 illustrate this comparison for two datasets: FashionMNIST and MNIST.

The graphs show that as the number of neurons increases, there is a corresponding increase in the total number of connections in the network. Initially, all connections are established, represented by the green curve (Initial Weights), which denotes the total number of weights before training begins. During training, the network undergoes structural changes where some connections are pruned, particularly those deemed less significant for the model's performance. This pruning process leads to the final state, represented by the orange curve (Final Weights), which shows a reduction in the total number of active connections.

The consistent gap between the initial and final weight curves suggests that pruning is an inherent aspect of the training dynamics, especially as the model grows in complexity with larger neuron sizes. This pruning helps in reducing model complexity, potentially leading to improved generalization and reduced computational costs without significantly compromising performance. The larger the network, the more potential there is for pruning, as seen by the increasing distance between the initial and final weight counts at higher neuron sizes.

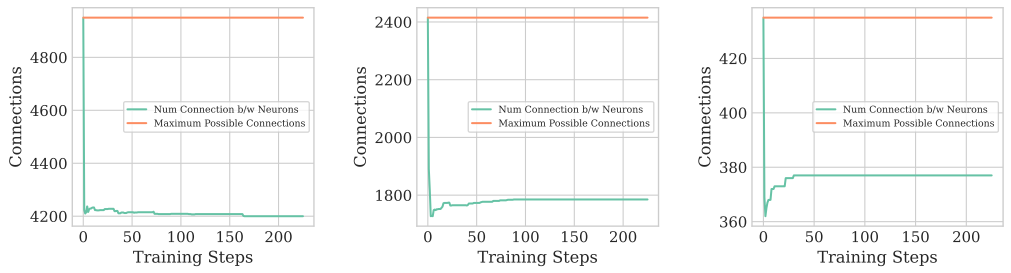


Figure 9: Number of connections i.e. weights in the network with training steps for (i) 100 Neurons (ii) 70 Neurons (iii) 30 Neurons

B Comparison of Initial and Final number of Weights with number of training steps

Figure 9 illustrates the impact of the FreeEvolve algorithm on the FreeNets architecture across different neuron sizes (100, 70, and 30 neurons). The plots compare the number of active connections between neurons throughout the training steps with the maximum possible connections within the network. At the beginning of the training, there is a noticeable drop in the number of active connections. This initial reduction can be attributed to overpruning caused by spurious activations, which occur due to the limited representation of the dataset in the early training phase. As the training progresses, the activations become a better indicator of the samples in the dataset, leading to stabilization in the number of active connections, as shown by the green curves.

The stabilization phase reflects the FreeEvolve algorithm's capacity to adapt more accurately to the dataset as a whole, fine-tuning the network architecture based on activations which are data driven. Relatively larger models, such as the ones with 100 neurons, experience a more substantial initial pruning but gradually stabilize, showcasing the adaptive nature of FreeEvolve. These observations highlight the algorithm's effectiveness in dynamically pruning unnecessary connections while maintaining those that contribute significantly to the model's performance, making the FreeNets architecture more efficient and robust over time.