GROWING NETWORKS BY FOLDING MANIFOLDS AT MISTAKES: APPENDIX

Anonymous authors
Paper under double-blind review

A APPENDIX - RELATED WORK

The problems of over-parameterization and lack of interpretability are so fundamental in deep learning that considerable research efforts have been devoted to addressing them separately.

A.1 OPTIMIZATION OF NEURAL NETWORK ARCHITECTURE

Various research directions try to optimize neural network architectures for efficient parameter utilization and address over-parameterization. Neural Architecture Search (NAS) is a prominent type of studies that automates the search for optimal neural network architectures within defined candidate structures. Examples include evolutionary methods like NEAT (Stanley & Miikkulainen, 2002), which mutates topologies via genetic algorithms, and meta-learning techniques (Elsken et al., 2020) that employ gradient-based optimization. However, NAS typically requires resource-intensive searches and may yield sub-optimal architectures due to the heuristic exploration of search spaces constrained by predefined operation sets or templates, given that exhaustive evaluation of the entire topology space is often infeasible.

Other approaches conduct "post-training" architecture optimization: Network pruning (Cheng et al., 2024) removes redundant parameters, while knowledge distillation (Gou et al., 2021) transfers knowledge to compact models. Both methods depend on pre-trained over-parameterized networks and do not inherently guarantee the most parameter-efficient architecture.

Dynamic Neural Networks (DyNNs) (Han et al., 2021) adapt computation graphs during inference in response to inputs (e.g., by adjusting depth, width, or routing paths). However, these methods mainly focus on hardware-friendly and computationally efficient architectures, and most operate at a macro-level granularity (e.g., adding or removing entire network blocks or modules), which limits their ability to achieve globally optimal parameter efficiency (Fayek et al., 2020).

In contrast to the above mainstream research directions, Growing Neural Networks (GrowNNs) propose to start with minimal architectures and incrementally grow their network structure during training. Unlike NAS, which searches massive candidate spaces starting from large, complex template structures, GrowNNs enable a streamlined uni-directional training process to obtain the final architecture without requiring heuristic or exhaustive searches. Furthermore, compared to post-training approaches (like network pruning and knowledge distillation) or inference-oriented techniques (like DyNNs), GrowNNs inherently avoid over-parameterized initializations. Therefore, GrowNNs offer a promising and theoretically efficient approach for achieving optimal architectures and mitigating over-parameterization.

GrowNNs is not a new direction: relevant research began even before the popularization of modern deep learning. Early works mainly proposed rules or algorithms to grow network graphs and topologies (e.g. the tiling algorithm (Mezard & Nadal, 1989), the tower algorithm (Gallant et al., 1990), the upstart algorithm (Frean, 1990), and others (Fahlman & Lebiere, 1989; Reilly et al., 1982; Platt, 1991)). These "classical" GrowNNs focused on automating structural increments, but did not relate structural changes to weight optimization, leaving the growing towards the optimal structures fundamentally a "trial-and-error" process. Research on GrowNNs was almost "paused" for a decade due to the emergence of modern deep learning, where researchers could simply initialize an over-parameterized model, train it on sufficient data, and achieve remarkable performance more readily than growing one from scratch. Recent breakthroughs in GrowNNs were achieved since 2019, when Wu et al. (2019) propose growing networks by splitting existing neurons into two symmetric copies with opposite perturbations. While this is an impressive idea, the method relies entirely on cur-

rent neurons and cannot directly introduce new neurons with significant perturbations for learning distinct features or insights. Evci et al. (2022) formulated the problem into specific questions of "when", "where" and "how" to grow networks, highlighting a key challenge of GrowNNs: how to initialize values for the newly added parameters after growth. They proposed the GradMax algorithm to address the "how" question by initializing the new parameters to maximize their gradient norm before subsequent training so as to prioritizes their updates relative to original parameters. This work was later extended by the follow-up studies, improving efficiency via variance transfer (Yuan et al., 2023; Verbockhaven et al., 2024) or template fusion (Pham et al., 2024). Notably, to show advantage in efficiency in order to compete with other approaches like NAS, all GrowNNs methods on GradMax grow multiple neurons simultaneously per step rather than adding a single neuron at a time. This growing strategy obscures individual neuron contributions and renders the process heuristic.

Nevertheless, nearly all neural architecture optimization approaches mentioned above are empirically driven based on intuitions. There is rarely well-established theory to justify whether the architectural adjustment really optimize towards most parameter-efficient architecture of the specific tasks nor interpretation on what such structural changes the models' expressive power or learning capacities.

A.2 eXplainable Artificial Intelligence (XAI) in Deep Learning

On the other hand, studies that attempt to open deep learning's "black box" problem fall under eXplainable AI (XAI) (Minh et al., 2022; Angelov et al., 2021; Samek et al., 2021). XAI methods are categorized using various taxonomies; a key taxonomy distinguishes them by their mechanism for achieving interpretability:

Ante-hoc XAI methods prioritize intrinsic interpretability within the model structure itself, thereby designing neural network modules or architectures that are inherently transparent or explainable. Attention mechanisms (e.g., Attention-based CNNs (Jetley et al., 2018) and Transformers (Vaswani et al., 2017)) dynamically weight input features to indicate the model's focus during prediction. Self-Explaining Networks (Alvarez-Melis & Jaakkola, 2018) enforce locally linear concept-based reasoning. Prototypical Networks (Chen et al., 2019) compare inputs to learned prototypes. Some others proposed architectures (e.g., Neural-Backed Decision Trees (Wan et al., 2020) and Concept Bottleneck Models (Koh et al., 2020)) or regularizations (e.g., sparsity (Louizos et al., 2018)) that are interpretable by design. For most of these ante-hoc methods, interpretability arises from representation disentanglement (Higgins et al., 2018), where the model's structure internally aligns learned features with human-understandable concepts during training. This design tends to make interpretability highly dependent on the specific model architecture, often sacrificing flexibility or performance for built-in explainability.

Post-hoc XAI methods apply explanation techniques to models after training is complete. Perturbation-based methods (e.g., LIME (Ribeiro et al., 2016) and SHAP (Lundberg & Lee, 2017)) probe the model by perturbing inputs and observing output changes. Gradient-based methods (e.g., Grad-CAM (Selvaraju et al., 2017)) trace gradient flow to determine the importance of inputs or features. Others try to uses rule-based (e.g., anchors (Ribeiro et al., 2018)) or activation-based (e.g., DeepLIFT (Shrikumar et al., 2017)) approximations for explaining network behaviors. Notably, most post-hoc methods perform only "attribution analysis" (e.g., identifying correlations or input features influencing outputs) rather than true "causal analysis" (which requires establishing cause-effect relationships): they only generate approximations, which may not always reflect the model's actual computations and thus should not be equated with inherently interpretable models.

In a nutshell, most existing XAI methods either constrain model architecture, sacrificing parameter efficiency (ante-hoc XAI) or may fail to deliver genuinely faithful and reliable interpretability (post-hoc XAI). Limited studies attempt to address both challenges of over-parameterization and lack of interpretability simultaneously. Therefore, developing an intrinsically explainable deep learning paradigm remains a significant unmet need.

B APPENDIX - PROOFS

B.1 PROOF OF EXISTENCE OF LEARNED MANIFOLD (SET) IN DEFINITION 1

The existence of learned manifolds for neural networks with piecewise linear activation functions (e.g., ReLU) has been established in prior work, which identified such structures as "response regions" (Pascanu et al., 2013), "linear regions" (Montúfar et al., 2014), and "activation patterns" (Raghu et al., 2017). In these cases, the existence of locally Euclidean substructures can be easily proved since the continuous regions in the input spaces that share the same "activation patterns" simply adopt the same affine transformation of the neural network after activation.

Here we generalize this existence proof to networks with smooth activations (e.g., sigmoid, tanh, softplus), extending the theoretical foundation beyond piecewise-linear regimes.

B.1.1 AFFECTING REGION AS A D-DIMENSIONAL MANIFOLD

The definition of learned manifold (set) $\mathcal{M}_\theta$ arises from the observation that for a deep neural network $f_\theta : \mathbb{R}^D \to \mathcal{Y}$, not all inputs in the entire domain $\mathbb{R}^D$ are meaningful. Instead, only specific regions of the input space induce effective behavior of f_θ for down-stream tasks, particularly for regression or classification.

Therefore, before proving the existence of learned manifold (set), we first define "affecting region" as an open set containing all these input-space regions where the network is smooth and task-relevant:

Definition 1 (Affecting Region)**.** *Let $\mathcal{M} \subset \mathbb{R}^D$ be a compact d-dimensional data manifold ($d \ll D$), and $\mathcal{X} \subset \mathbb{R}^D$ a finite set of training samples from a neighborhood of $\mathcal{M}$. For a trained neural network $f_\theta : \mathbb{R}^D \to \mathcal{Y}$, the affecting region $\mathcal{A} \subset \mathbb{R}^D$ is an open set containing:*

1. *The convex hull of the training set $\mathcal{X}$,*
2. *The data manifold $\mathcal{M}$,*
3. *All paths in $\mathbb{R}^D$ between points in $\mathcal{X}$ along which f_θ is C^k-smooth ($k \geq 1$).*

which has the following key properties:

1. *Since $\mathcal{M}$ is compact and $\mathcal{X}$ is finite, $\mathcal{A}$ is bounded,*
2. *For smooth activations (e.g., sigmoid, softplus), f_θ is C^k-smooth on an open neighborhood of $\mathcal{M}$, thus $\mathcal{A}$ exists as an open set containing $\mathcal{M}$ and $\mathcal{X}$,*
3. *As an open subset of $\mathbb{R}^D$, $\mathcal{A}$ is locally Euclidean: $\forall x \in \mathcal{A}, \exists B_\epsilon(x) \subset \mathcal{A}$ homeomorphic to $\mathbb{R}^D$.*

Thus, $\mathcal{A}$ is a topological manifold with boundary of dimension D.

Now, we can prove the learned manifold (set) $\mathcal{M}_\theta$ as a sub-manifold of $\mathcal{A}$ for specific downstream tasks of regression and classification.

B.1.2 PROOF OF LEARNED MANIFOLD (SET) FOR REGRESSION TASKS

Let $f_\theta = \psi \circ \phi$ be a trained neural network approximating $g : \mathcal{M} \to \mathcal{Y}$. For regression tasks, the learned manifold $\mathcal{M}\theta$ is defined as a collection of connected topological manifolds in regions where ϕ is locally injective, satisfying $\phi(\mathcal{M}\theta) \approx \phi(\mathcal{M})$.

By the Manifold Learning Theorem, ϕ is an immersion on $\mathcal{M}$ with $\text{rank}(D\phi) \equiv d$. Since $\mathcal{M}$ is compact, for each $x \in \mathcal{M}$, there exists an open set $U_x \subset \mathcal{A}$ where ϕ has constant rank d.

By the Constant Rank Theorem (Tu, 2011), there exist local coordinates (V_x, β_x) around x and (W_x, γ_x) around $\phi(x)$ such that:

$$\gamma_x \circ \phi \circ \beta_x^{-1}(x_1, ..., x_D) = (x_1, x_2, ..., x_d, 0, ..., 0)$$

.

Thus, $\phi(U_x)$ is a d-dimensional embedded submanifold of $\mathbb{R}^h$.

For each U_x, define a local section $\alpha_x : \mathbb{R}^d \to U_x$ by fixing $c_{d+1}, \ldots, c_D$ (e.g., coordinates of x):

$$\alpha_x(x_1, ..., x_d) = \beta_x^{-1}(x_1, x_2, ..., x_d, c_{d+1}, ..., c_D)$$

.

The image $\Sigma_x = \alpha_x(\mathbb{R}^d) \cap \phi^{-1}(\phi(\mathcal{M} \cap U_x))$ is an embedded d-dimensional submanifold satisfying $\phi(\Sigma_x) = \phi(\mathcal{M} \cap U_x)$.

Construct $\mathcal{S} = \bigcup_{x \in \mathcal{M}} \Sigma_x \subset \mathcal{A}$. Then:

1. $\mathcal{S}$ is a d-dimensional topological manifold (as local charts agree on overlaps).
2. $\phi(\mathcal{S}) = \phi(\mathcal{M})$ (by construction).
3. ϕ is locally injective on $\mathcal{S}$ (since rank$(D\phi) = d$).

Let $\{\mathcal{M}\theta^{(k)}\}_{k=1}^K$ be the connected components of $\mathcal{S}$. Each $\mathcal{M}_\theta^{(k)}$ is a connected topological manifold, and $\phi(\mathcal{M}_\theta^{(k)}) \subseteq \phi(\mathcal{M})$. By continuity of ϕ and compactness of $\mathcal{M}$, $\phi(\mathcal{M}_\theta^{(k)}) \approx \phi(\mathcal{M})$.

Thus, $\mathcal{M}_\theta = \bigcup_{k=1}^K \mathcal{M}_\theta^{(k)}$ is the learned manifold.

B.1.3 Proof of Learned Manifold (Set) for Classification Tasks

In a $\mathcal{K}$-classes classification task, for two classes $a, b \in \mathcal{K}$, define $\mathcal{M}_a = x \in \mathcal{M} : x$ class a and $\mathcal{M}_b$ analogously. Let $f_\theta = \psi \circ \phi$ be the trained network. The decision boundary between classes a and b is:

$$\mathcal{B}_{a,b} = \{x \in \mathcal{A} : f_\theta(x)[a] = f_\theta(x)[b]\}$$

.

Let the output layer ψ in the Manifold Learning Theorem be a linear function:

$$\mathcal{B}_{a,b} = \{x \in \mathcal{A} : \mathbf{w}^\top \phi(x) + b = 0\}$$

,

where $\mathbf{w} \in \mathbb{R}^h$, $b \in \mathbb{R}$ are final-layer parameters.

Define $h_{a,b} : \mathbb{R}^D \to \mathbb{R}$ as $h_{a,b}(x) = \mathbf{w}^\top \phi(x) + b$. Since ϕ is C^k on $\mathcal{A}$ ($k \geq 1$), $h_{a,b}$ is C^k-smooth. The regular set is:

$$\mathcal{R}_{a,b} = \{x \in \mathcal{B}_{a,b} : Dh_{a,b}(x) \neq 0\}$$

.

$\mathcal{R}a, b$ is open in $\mathcal{B}a, b$. By the Preimage Theorem (Guillemin & Pollack, 2010), $\mathcal{R}_{a,b}$ is a $(D-1)$-dimensional C^k embedded submanifold of $\mathbb{R}^D$.

Let $\mathcal{M}\theta : a, b^{(j)} j = 1^J$ be the connected components of $\mathcal{R}a, b$. Each $\mathcal{M}\theta : a, b^{(j)}$ is a connected $(D-1)$-dimensional topological manifold and:

1. $\mathcal{M}\theta : a, b^{(j)} \subset \mathcal{B}a, b$ (by construction),
2. $\phi(\mathcal{M}\theta : a, b^{(j)}) \subset Ha, b = z \in \mathbb{R}^h : \mathbf{w}^\top z + b = 0$ (since $h_{a,b}(x) = 0$).

Training ensures $h_{a,b}(x) > 0$ for $x \in \mathcal{M}a$ and $ha, b(x) < 0$ for $x \in \mathcal{M}b$, so Ha, b linearly separates $\phi(\mathcal{M}_a)$ and $\phi(\mathcal{M}_b)$.

Non-emptiness follows from:

- $\mathcal{M}_a, \mathcal{M}_b \neq \emptyset$ (data assumption),
- Path-connectedness of $\mathcal{A}$ and intermediate value theorem ensure $\mathcal{B}_{a,b} \neq \emptyset$,
- Sard's Theorem implies $\mathcal{R}_{a,b} \neq \emptyset$.

Thus, $\mathcal{M}\theta : a, b = \bigcup j = 1^{J} \mathcal{M}^{(j)}_{\theta:a,b}$ is the learned manifold.

By continuity and the universal approximation properties of deep networks, there exists a compact neighborhood $\mathcal{N} \subset \mathcal{A}$ of $\mathcal{M}$ where f_θ maintains its approximating behavior.

Consider the mapping $\phi|_{\mathcal{N}} : \mathcal{N} \to \mathbb{R}^h$. By the constant rank theorem (Tu, 2011), for any point $x \in \mathcal{N}$ where the Jacobian $D\phi(x)$ has constant rank r in a neighborhood of x, there exist local coordinates in which ϕ acts as a linear projection. The set of points where $D\phi(x)$ has maximal rank is open and dense in $\mathcal{N}$ for smooth functions.

Define the regression surface $\mathcal{M}_\theta$ as the union of connected components of the set:

$$\{x \in \mathcal{N} : rank(D\phi(x)) = d\}$$

that intersect $\mathcal{M}$, where we take the closure of these components within $\mathcal{N}$.

Manifold structure: For each $x \in \mathcal{M}_\theta$ with rank$(D\phi(x)) = d$, by the constant rank theorem, there exists a neighborhood $U_x \subset \mathcal{N}$ such that $\phi(U_x)$ is a d-dimensional embedded submanifold of $\mathbb{R}^h$. The preimage $\phi^{-1}(\phi(U_x)) \cap U_x$ forms a d-dimensional submanifold of $\mathbb{R}^D$ near x.

Approximation property: Since $f_\theta \approx g$ on $\mathcal{M}$ and ψ is simple (e.g., linear), the network must learn representations such that $\phi(\mathcal{M})$ is approximately unfolded in $\mathbb{R}^h$. The regression surface $\mathcal{M}\theta$ is constructed to satisfy $\phi(\mathcal{M}\theta) \approx \phi(\mathcal{M})$ by including precisely the connected components that preserve the dimensional and topological structure of $\mathcal{M}$ under ϕ.

Local injectivity: In regions where rank$(D\phi(x)) = d$, the mapping ϕ is locally injective, ensuring that $\mathcal{M}_\theta$ captures the regression surface without redundant degrees of freedom.

Therefore, $\mathcal{M}_\theta$ exists as a collection of d-dimensional connected sub-manifolds satisfying the definition.

B.2 Proof of Theorem 2

We prove the two cases separately by analyzing how adding a neuron affects the network's functional representation and consequently the geometry of the learned manifold.

B.2.1 Non-Smooth Folding (Piecewise Linear Activations)

Consider a deep neural network $f_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma_{L-1}\Big(\mathbf{W}_{L-1}\sigma_{L-2}\big(\cdots\sigma_1(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1)\cdots\big)+\mathbf{b}_{L-1}\Big)$ with piecewise linear activation σ (e.g., ReLU). Its learned manifold $\mathcal{M}\boldsymbol{\theta}$ is piecewise linear, consisting of different regions of affine transformations.

As established in **?**, each neuron contributes to partitioning the input space into linear regions. When we add a neuron to layer i, the pre-activation pattern gains an additional dimension, and the post-activation through σ introduces a new hyperplane boundary in the feature space of layer i that subdivides existing linear regions.

This hyperplane boundary propagates through subsequent layers, creating additional subdivisions in the input space. By the compositional structure of deep networks, each subdivision in layer i can induce multiple subdivisions in the input space (**?**). Consequently, the number of maximal connected regions where $f_{\boldsymbol{\theta}}$ is affine increases:

$$\mathcal{N}(\mathcal{M}_{\theta'}) = \mathcal{N}(\mathcal{M}_\theta) + \Delta\mathcal{N}$$

where $\Delta\mathcal{N} > 0$ represents the additional regions created by the new neuron's activation pattern.

The manifold $\mathcal{M}_{\theta'}$ thus has strictly greater combinatorial complexity, with more piecewise affine segments and additional boundaries (non-differentiable edges where the function's derivatives are discontinuous) compared to $\mathcal{M}_{\boldsymbol{\theta}}$. Geometrically, these boundaries correspond to "creases" on the learned manifold $\mathcal{M}_{\boldsymbol{\theta}'}$.

The smoothness measure $\mathcal{C}$, which counts the number of continuous derivatives, decreases because the new manifold contains additional points where the function fails to be differentiable. Formally,

if the original manifold had smoothness class $\mathcal{C}^k$ and we introduce creases (where only $\mathcal{C}^0$ continuity holds), then:

$$\mathcal{C}(\mathcal{M}_{\boldsymbol{\theta}'}) \leq \mathcal{C}^0 < \mathcal{C}^k \leq \mathcal{C}(\mathcal{M}_{\boldsymbol{\theta}})$$

B.2.2 Smooth Folding (Smooth Activations)

For smooth activations $\sigma \in C^\infty$ (e.g., sigmoid, tanh), the network $f_{\boldsymbol{\theta}}$ is smooth, and the decision surface $\mathcal{M}_{\boldsymbol{\theta}}$ is a smooth manifold (possibly with boundary).

Adding a neuron increases the network's functional complexity without introducing discontinuities. The new network $f_{\boldsymbol{\theta}'}$ can represent more complex functions (**??**), which translates to increased geometric complexity of the level set $\{\mathbf{x} : f_{\boldsymbol{\theta}'}(\mathbf{x}) = 0\}$.

The squared curvature norm $\|\kappa\|^2$ provides a natural measure of geometric complexity that is invariant to parameterization and captures both principal curvatures. Since smooth activations allow the network to approximate functions with arbitrarily high curvature (**?**), adding neurons enables the learning of manifolds with greater curvature variation.

Formally, the curvature integral increases because:

1. The additional neuron increases the network's capacity to represent oscillatory functions
2. The smoothness of σ ensures the curvature is well-defined everywhere
3. The increased parameter space allows the network to fit more complex decision boundaries while maintaining the smoothness constraint

Therefore, for the optimally trained networks representing the maximum-complexity manifolds learnable by each architecture:

$$\int_{\mathcal{M}_{\boldsymbol{\theta}'}} \|\kappa\|^2 dV \geq \int_{\mathcal{M}_{\boldsymbol{\theta}}} \|\kappa\|^2 dV$$

with strict inequality holding generically when the additional capacity is utilized.

B.3 Proof of Theorem 3

Consider the effect of adding a new neuron to the i-th layer of the neural network f_θ. The network's computation up to layer i can be expressed as:

$$\phi_i(\mathbf{x}) = \sigma_i\left(\mathbf{W}i\phi i - 1(\mathbf{x}) + \mathbf{b}_i\right)$$

where $\phi_{i-1} : \mathbb{R}^D \to \mathbb{R}^{d_{i-1}}$ is the feature map from the input to layer $i-1$.

After growing a neuron, the new weight matrix and bias vector for layer i become:

$$\mathbf{W}i^{\text{new}} = [\mathbf{W}i\ \mathbf{w}i^{\text{new}}], \quad \mathbf{b}i^{\text{new}} = [\mathbf{b}i\ bi^{\text{new}}] \tag{1}$$

The pre-activation of the new neuron for input $\mathbf{x}$ is:

$$z^{\text{new}}(\mathbf{x}) = \langle \mathbf{w}i^{\text{new}}, \phi i - 1(\mathbf{x}) \rangle + b_i^{\text{new}}$$

The critical observation is that the activation function σ_i (whether piecewise linear like ReLU or smooth like tanh) exhibits its most significant non-linear behavior precisely where its input transitions through zero. Specifically:

- For **piecewise linear activations** (e.g., ReLU): The function is non-differentiable at zero, creating a sharp "crease" in the learned manifold.
- For **smooth activations** (e.g., sigmoid, tanh): The function has maximum curvature near zero, inducing smooth "bending" of the manifold.

The set where this non-linear transformation occurs is defined by:

$$\mathcal{F} = \mathbf{x} \in \mathbb{R}^D \mid z^{\text{new}}(\mathbf{x}) = 0 \qquad = \mathbf{x} \in \mathbb{R}^D \mid \langle \mathbf{w}i^{\text{new}}, \phi i - 1(\mathbf{x}) \rangle + b_i^{\text{new}} = 0$$

This represents a $(D-1)$-dimensional surface in the input space $\mathbb{R}^D$ (or a surface of co-dimension 1) where the folding operation manifests. The geometry of $\mathcal{F}$ is determined entirely by the parameters $\mathbf{w}i^{\text{new}}$ and bi^{new}, along with the feature map ϕ_{i-1}.

The output-weight vector $\mathbf{w}_{i+1}^{\text{new}}$ scales the contribution of the new neuron to the next layer but does not affect the *location* of the folding surface $\mathcal{F}$. It only influences the *magnitude* and *direction* of the folding in the output space, not its geometric locus in the input space.

Therefore, the folding of the learned manifold occurs precisely on the hyper-surface $\mathcal{F}$ defined by the pre-activation condition of the new neuron, and this location is independent of the output-weight vector $\mathbf{w}_{i+1}^{\text{new}}$.

B.4 Proof of Theorem 4

Let f_θ be the original neural network and f_θ^{new} be the network after adding a neuron to the i-th layer. The key observation is that all network computations remain identical up to layer $i-1$, producing the feature map $\phi_{i-1}(\mathbf{x})$.

In the original network, the i-th layer computes:

$$\phi_i(\mathbf{x}) = \sigma(W_i \phi_{i-1}(\mathbf{x}) + \mathbf{b}_i)$$

where W_i is the weight matrix and $\mathbf{b}_i$ the bias vector of layer i.

After adding the new neuron, the i-th layer weight matrix expands to:

$$W_i^{\text{new}} = \begin{bmatrix} W_i \\ \mathbf{w}_i^{\text{new}} \end{bmatrix}, \quad \mathbf{b}_i^{\text{new}} = \begin{bmatrix} \mathbf{b}_i \\ b_i^{\text{new}} \end{bmatrix}$$

The new i-th layer activation becomes:

$$\phi_i^{\text{new}}(\mathbf{x}) = \sigma\left(\begin{bmatrix} W_i \\ \mathbf{w}_i^{\text{new}} \end{bmatrix} \phi_{i-1}(\mathbf{x}) + \begin{bmatrix} \mathbf{b}_i \\ b_i^{\text{new}} \end{bmatrix} \right) = \begin{bmatrix} \phi_i(\mathbf{x}) \\ \sigma(\mathbf{w}_i^{\text{new}} \cdot \phi_{i-1}(\mathbf{x}) + b_i^{\text{new}}) \end{bmatrix}$$

The $(i+1)$-th layer weight matrix expands accordingly to accommodate the new input dimension:

$$W_{i+1}^{\text{new}} = \begin{bmatrix} W_{i+1} & \mathbf{w}_{i+1}^{\text{new}} \end{bmatrix}$$

where W_{i+1} are the original weights and $\mathbf{w}_{i+1}^{\text{new}}$ are the outgoing weights from the new neuron.

The output of layer $i+1$ in the new network is:

$$\phi_{i+1}^{\text{new}}(\mathbf{x}) = W_{i+1}^{\text{new}} \phi_i^{\text{new}}(\mathbf{x}) = W_{i+1} \phi_i(\mathbf{x}) + \mathbf{w}_{i+1}^{\text{new}} \cdot \sigma(\mathbf{w}_i^{\text{new}} \cdot \phi_{i-1}(\mathbf{x}) + b_i^{\text{new}})$$

Since all subsequent layers $(i+2)$ and beyond remain unchanged, and their computations depend only on $\phi_{i+1}^{\text{new}}(\mathbf{x})$, the total output displacement is:

$$\Delta f_\theta = f_\theta^{\text{new}} - f_\theta = \mathbf{w}_{i+1}^{\text{new}} \cdot \sigma(\mathbf{w}_i^{\text{new}} \cdot \phi_{i-1}(\mathbf{x}) + b_i^{\text{new}})$$

This shows that $\mathbf{w}_{i+1}^{\text{new}}$ governs the output displacement direction (as it projects the neuron's activation to the output space), while $\mathbf{w}_i^{\text{new}}$ and b_i^{new} control the magnitude through the activation function σ.

B.5 Proof of Theorem 5

Let $L(f_\theta(\mathbf{x}), \mathbf{y})$ be the loss function. The input-space gradient of the original network is:

$$\frac{\partial L(f_\theta(\mathbf{x}), \mathbf{y})}{\partial \mathbf{x}} = \frac{\partial L}{\partial f_\theta} \cdot \frac{\partial f_\theta}{\partial \mathbf{x}}$$

where $\frac{\partial L}{\partial f_\theta}$ is the gradient of the loss with respect to the network output.

From the previous theorem, the new network output is:

$$f_\theta^{\text{new}}(\mathbf{x}) = f_\theta(\mathbf{x}) + \Delta f_\theta(\mathbf{x}) = f_\theta(\mathbf{x}) + \mathbf{w}_{i+1}^{\text{new}} \cdot \sigma(\mathbf{w}_i^{\text{new}} \cdot \phi_{i-1}(\mathbf{x}) + b_i^{\text{new}})$$

The gradient perturbation in input space is:

$$\Delta \frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L(f_\theta^{\text{new}}(\mathbf{x}), \mathbf{y})}{\partial \mathbf{x}} - \frac{\partial L(f_\theta(\mathbf{x}), \mathbf{y})}{\partial \mathbf{x}}$$

Using the chain rule and noting that the perturbation is small, we approximate:

$$\frac{\partial L(f_\theta^{\text{new}}(\mathbf{x}), \mathbf{y})}{\partial \mathbf{x}} \approx \frac{\partial L(f_\theta(\mathbf{x}), \mathbf{y})}{\partial f_\theta(\mathbf{x})} \cdot \frac{\partial f_\theta^{\text{new}}(\mathbf{x})}{\partial \mathbf{x}}$$

The gradient of the new output with respect to the input is:

$$\frac{\partial f_\theta^{\text{new}}(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}} + \frac{\partial \Delta f_\theta(\mathbf{x})}{\partial \mathbf{x}}$$

Focusing on the perturbation term:

$$\frac{\partial \Delta f_\theta(\mathbf{x})}{\partial \mathbf{x}} = \frac{\partial}{\partial \mathbf{x}} \left[\mathbf{w}_{i+1}^{\text{new}} \cdot \sigma(\mathbf{w}_i^{\text{new}} \cdot \phi_{i-1}(\mathbf{x}) + b_i^{\text{new}}) \right]$$

Applying the chain rule:

$$\frac{\partial \Delta f_\theta(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{w}_{i+1}^{\text{new}} \cdot \sigma'(s) \cdot \frac{\partial}{\partial \mathbf{x}} \left[\mathbf{w}_i^{\text{new}} \cdot \phi_{i-1}(\mathbf{x}) + b_i^{\text{new}} \right]$$

where $s = \mathbf{w}_i^{\text{new}} \cdot \phi_{i-1}(\mathbf{x}) + b_i^{\text{new}}$.

Since b_i^{new} is constant with respect to $\mathbf{x}$, its derivative vanishes:

$$\frac{\partial \Delta f_\theta(\mathbf{x})}{\partial \mathbf{x}} = \mathbf{w}_{i+1}^{\text{new}} \cdot \sigma'(s) \cdot \frac{\partial}{\partial \mathbf{x}} \left[\mathbf{w}_i^{\text{new}} \cdot \phi_{i-1}(\mathbf{x}) \right]$$

The derivative of the linear transformation is:

$$\frac{\partial}{\partial \mathbf{x}} \left[\mathbf{w}_i^{\text{new}} \cdot \phi_{i-1}(\mathbf{x}) \right] = \mathbf{J}_{\phi_{i-1}}(\mathbf{x})^\top \mathbf{w}_i^{\text{new}}$$

where $\mathbf{J}_{\phi_{i-1}}(\mathbf{x})$ is the Jacobian matrix of ϕ_{i-1} with respect to $\mathbf{x}$.

Combining these results, the gradient perturbation becomes:

$$\Delta \frac{\partial L}{\partial \mathbf{x}} = \left(\frac{\partial L(f_\theta(\mathbf{x}), \mathbf{y})}{\partial f_\theta(\mathbf{x})} \cdot \mathbf{w}_{i+1}^{\text{new}} \right) \sigma'(s) \cdot \mathbf{J}_{\phi_{i-1}}(\mathbf{x})^\top \mathbf{w}_i^{\text{new}}$$

In the activated region where $s > 0$ and $\sigma'(s) > 0$ (e.g., ReLU with $\sigma'(s) = 1$ for $s > 0$):

- The direction of the gradient perturbation is determined by $\mathbf{J}_{\phi_{i-1}}(\mathbf{x})^\top \mathbf{w}_i^{\text{new}}$, which depends on $\mathbf{w}_i^{\text{new}}$ and the feature mapping Jacobian.

- The magnitude is scaled by $\frac{\partial L}{\partial f_\theta} \cdot \mathbf{w}_{i+1}^{\text{new}}$, with $\mathbf{w}_{i+1}^{\text{new}}$ acting as a scalar multiplier.

- The bias b_i^{new} affects the activation threshold but not the gradient direction or magnitude once the neuron is active.

This demonstrates that $\mathbf{w}_i^{\text{new}}$ governs the input-space gradient direction through its interaction with the feature mapping Jacobian.

REFERENCES

David Alvarez-Melis and Tommi S Jaakkola. Self-explaining neural networks. *SENN. pdf*, 2018.

Plamen P Angelov, Eduardo A Soares, Richard Jiang, Nicholas I Arnold, and Peter M Atkinson. Explainable artificial intelligence: an analytical review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 11(5):e1424, 2021.

Chaofan Chen, Oscar Li, Alina Barnett, Jonathan Su, and Cynthia Rudin. This looks like that: Deep learning for interpretable image recognition. *NeurIPS*, 32, 2019.

Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

Thomas Elsken, Benedikt Staffler, Jan Hendrik Metzen, and Frank Hutter. Meta-learning of neural architectures for few-shot learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12365–12375, 2020.

Utku Evci, Bart van Merrienboer, Thomas Unterthiner, Max Vladymyrov, and Fabian Pedregosa. Gradmax: Growing neural networks using gradient information. *arXiv preprint arXiv:2201.05125*, 2022.

Scott Fahlman and Christian Lebiere. The cascade-correlation learning architecture. *Advances in neural information processing systems*, 2, 1989.

Haytham M Fayek, Lawrence Cavedon, and Hong Ren Wu. Progressive learning: A deep learning framework for continual learning. *Neural Networks*, 128:345–357, 2020.

Marcus Frean. The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural computation*, 2(2):198–209, 1990.

Stephen I Gallant et al. Perceptron-based learning algorithms. *IEEE Transactions on neural networks*, 1(2):179–191, 1990.

Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International journal of computer vision*, 129(6):1789–1819, 2021.

Victor Guillemin and Alan Pollack. *Differential topology*, volume 370. American Mathematical Soc., 2010.

Yizeng Han, Gao Huang, Shiji Song, Le Yang, Honghui Wang, and Yulin Wang. Dynamic neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(11): 7436–7456, 2021.

Irina Higgins, David Amos, David Pfau, Sebastien Racaniere, Loic Matthey, Danilo Rezende, and Alexander Lerchner. Towards a definition of disentangled representations. *arXiv preprint arXiv:1812.02230*, 2018.

Saumya Jetley, Nicholas A Lord, Namhoon Lee, and Philip HS Torr. Learn to pay attention. *arXiv preprint arXiv:1804.02391*, 2018.

Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *International conference on machine learning*, pp. 5338–5348. PMLR, 2020.

Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through L_0 regularization. *ICLR*, 2018.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in Neural Information Processing Systems*, pp. 4765–4774, 2017.

Marc Mezard and Jean-P Nadal. Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics A: Mathematical and General*, 22(12):2191, 1989.

Dang Minh, H Xiang Wang, Y Fen Li, and Tan N Nguyen. Explainable artificial intelligence: a comprehensive review. *Artificial Intelligence Review*, 55(5):3503–3568, 2022.

Guido Montúfar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *Advances in neural information processing systems*, 27, 2014.

Razvan Pascanu, Guido Montufar, and Yoshua Bengio. On the number of response regions of deep feed forward networks with piece-wise linear activations. *arXiv preprint arXiv:1312.6098*, 2013.

Chau Pham, Piotr Teterwak, Soren Nelson, and Bryan A Plummer. Mixturegrowth: Growing neural networks by recombining learned parameters. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 2800–2809, 2024.

John Platt. A resource-allocating network for function interpolation. *Neural computation*, 3(2): 213–225, 1991.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl-Dickstein. On the expressive power of deep neural networks. In *international conference on machine learning*, pp. 2847–2854. PMLR, 2017.

Douglas L Reilly, Leon N Cooper, and Charles Elbaum. A neural model for category learning. *Biological cybernetics*, 45(1):35–41, 1982.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144, 2016.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations, 2018.

Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J Anders, and Klaus-Robert Müller. Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings of the IEEE*, 109(3):247–278, 2021.

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 618–626, 2017.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pp. 3145–3153. PMLR, 2017.

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

Loring W Tu. Manifolds. In *An Introduction to Manifolds*, pp. 47–83. Springer, 2011.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Manon Verbockhaven, Théo Rudkiewicz, Sylvain Chevallier, and Guillaume Charpiat. Growing tiny networks: Spotting expressivity bottlenecks and fixing them optimally. *Transactions on Machine Learning Research Journal*, 2024.

Alvin Wan, Lisa Dunlap, Daniel Ho, Jihan Yin, Scott Lee, Henry Jin, Suzanne Petryk, Sarah Adel Bargal, and Joseph E Gonzalez. Nbdt: Neural-backed decision trees. *arXiv preprint arXiv:2004.00221*, 2020.

Lemeng Wu, Dilin Wang, and Qiang Liu. Splitting steepest descent for growing neural architectures. *Advances in neural information processing systems*, 32, 2019.

Xin Yuan, Pedro Savarese, and Michael Maire. Accelerated training via incrementally growing neural networks using variance transfer and learning rate adaptation. *Advances in Neural Information Processing Systems*, 36:16673–16692, 2023.