Computing Linear Regions in Neural Networks with Skip Connections

Johnny Joyce Jan Verschelde

University of Illinois at Chicago
Department of Mathematics, Statistics, and Computer Science
851 S. Morgan St. (m/c 249), Chicago, IL, 6067-7045

Abstract

Neural networks are important tools in machine learning. Representing piecewise linear activation functions with tropical arithmetic enables the application of tropical geometry. Algorithms are presented to compute regions where the neural networks are linear maps. Through computational experiments, we provide insights on the difficulty to train neural networks, in particular on the problems of overfitting and on the benefits of skip connections.

Keywords. linear regions, neural networks, skip connections.

1 Introduction

A neural network is a composition of neurons, where each neuron can be represented as a nonlinear function depending on inputs and parameters, called weights and biases. The nonlinearity of the network can be understood via tropical geometry, in particular for networks with ReLU activation functions, which are piecewise linear. For such networks, we introduce an algorithm to compute all linear regions of a neural network. A linear region of a neural network is a connected region on which the map defined by the network is linear. Knowing those linear regions allows for quicker predictions, as demonstrated by our new caching algorithm. Our algorithms work for networks with skip connections. Skip connections add the output of previous layers to the input of later layers, skipping over the layers in between.

The expository paper [2] offers promising avenues to study neural networks. The number of linear regions in neural networks was explored in [12]

and shown in [16] to be exponential in the number number of layers, with more results in [14], though the actual number of linear regions has been observed to be much smaller in practice, growing linearly in the number of neurons [4]. The geometry and characteristics of linear regions was observed when training networks with different optimization techniques in [17]. Tropical geometry was applied to machine learning models in [10]. Linear regions have been studied through various perspectives, such as; by viewing regions as the result of "foldings" of space [12], by viewing regions as affine spline mappings [5] (which relies on a view of neural networks put forth in [1]), and by viewing regions as subsets of tropical maps [16]. The perspective we take in this paper is that of tropical maps.

The mentioned existing literature has studied linear regions in certain cases, but has not found exact bounds for *all* linear regions in any given network. As such, the problem we solve in this paper is to compute these exact bounds of all linear regions in any trained neural network with piecewise activation functions. Furthermore, we extend those algorithms to networks with skip connections and obtain insights into how skip connections produce their beneficial results.

These questions are similar to well-explored problems in computational geometry and combinatorial geometry, where one may examine properties of hyperplane arrangements, such as the set of all subspaces given by intersections of some hyperplanes (this set is called the intersection semilattice) [3]. However, a key difference in our problem is that these hyperplanes depend on the input given to the network because each layer is a function of all layers that precede it. Certain inputs may cause some combination of neurons to "fire" (in the sense that their post-activation function is positive) on the first layer, while other inputs may lead to other neurons firing, and the two cases will produce different pre-activation and post-activation values on all subsequent layers. Our problem is also similar to *binary space partitioning*, where half-spaces are recursively defined separately for each side of some previous hyperplanes, resulting in a tree structure [15]. However, a key difference is that each layer of a neural network introduces multiple hyperplanes at the same time (one for each neuron), resulting in tree nodes with more than two children.

In this paper, we use tropical geometry to explore linear regions in neural networks. In Sect. 2, we formulate and present recurrence relations that can be used to find hyperplanes that bound each linear region of any given network. Using these results, we present an algorithm to find the exact boundaries of every linear region in neural networks. A walkthrough of our algorithm is presented in Sect. 3. We also extend these results and algorithms

to networks with arbitrarily many skip connections in Sect. 4. In Sect. 5, we describe the outcomes of our computational experiments and visualizations, observing how patterns of the regions follow patterns in the training data. Finally, in Sect. 7, we present an algorithm for caching functions produced by neural networks on any given linear region, allowing for quicker predictions on unseen data. This algorithm was conceptualized in our previous paper [7].

All code used in this paper for running algorithms, generating figures, and performing statistical tests is publicly available at [8]. Most of the material for this paper originated in [6, Chapter 4].

2 Linear Regions of Neural Networks

In this section, we use tropical geometry to introduce an algorithm for finding all linear regions of a neural network.

Theorem 5.4 of L. Zhang et al. [16] shows that neural networks with piecewise linear activation functions are equivalent to tropical rational functions. They achieve this result by showing that if the inputs to the ℓ-th layer of the network are tropical rational functions, then the output of the layer is also a tropical rational function (Proposition 5.1 of [16], which we also use in later sections). Repeated application of this fact gives us Theorem 5.4 of [16], which states the equivalence of neural networks and tropical rational functions.

Consider a nonlinear neural network where nonlinearity is introduced only through ReLU activation functions. This assumption is mild, since more complex piecewise linear functions can be achieved by adding linear combinations of ReLU functions. For example, we have $\texttt{LeakyReLU}(x, a) = \texttt{ReLU}(x) - a \cdot \texttt{ReLU}(-x)$, where a specifies the gradient for inputs $x < 0$. With this assumption in mind, we have that the output of the first layer of a neural network is given by:

$$\nu(x) = \texttt{ReLU}(Ax + b) = \max(Ax + b, 0), \tag{1}$$

where A is the weight matrix of the first layer, and b is the corresponding bias. [16] shows that we decompose the matrix A as the difference of two matrices, $A := A_+ - A_-$, where the (i, j)-th entry of A_+ is $\max\{a_{ij}, 0\}$ and where the (i, j)-th entry of A_- is $\max\{-a_{ij}, 0\}$. This means that the layer $\nu(x)$ can be expressed as the difference of two tropical polynomials:

$$\begin{aligned} \nu(x) &= \max\{A_+x + b, A_-x\} - A_-x & (2)\\ &= \left((x^{\otimes A_+} \otimes b) \oplus x^{\otimes A_-}\right) \oslash A_-x & (3) \end{aligned}$$

where

$$x \oplus y = \max\{x, y\}, \quad x \otimes y = x + y, \quad x \oslash y = x + (-y) = x - y. \tag{4}$$

By recursively applying this decomposition of layers, [16] gives the following result, which we use as our starting point:

Proposition 2.1 (Proposition 5.1 of L. Zhang et al., 2018 [16]) *Let $A \in \mathbb{Z}^{n_{l+1} \times n_l}$, $b \in \mathbb{R}^{n_{l+1}}$ be the parameters of the $(\ell+1)$-th layer of a neural network with piecewise linear activation functions. Let $\rho^{(\ell)}$ and $\nu^{(\ell)}$ be the pre-activation and post-activation output functions of the ℓ-th layer, respectively. If the nodes of the ℓ-th layer are given by tropical rational functions,*

$$\nu^{(\ell)} = F^{(\ell)}(x) \oslash G^{(\ell)}(x) = F^{(\ell)}(x) - G^{(\ell)}(x) \tag{5}$$

i.e., each coordinate of $F^{(\ell)}$ and $G^{(\ell)}$ is a tropical polynomial in x, then the outputs of the pre-activation ($\rho^{(\ell+1)} \circ \nu^{(\ell)}$) and of the $(\ell+1)$-th layer ($\nu^{(\ell+1)}$) are given by the tropical rational functions

$$\begin{aligned} \rho^{(\ell+1)} \circ \nu^{(\ell)}(x) &= H^{(\ell+1)(x)} - G^{(\ell+1)}(x), & (6) \\ \nu^{(\ell+1)}(x) = \sigma \circ \rho^{(\ell+1} \circ \nu^{(\ell)}(x) &= F^{(\ell+1)(x)} - G^{(\ell+1)}(x), & (7) \end{aligned}$$

respectively, where

$$\begin{aligned} F^{(\ell+1)}(x) &= \max\{H^{(\ell+1)}(x), G^{(\ell+1)}(x)\}, & (8) \\ G^{(\ell+1)}(x) &= A_+ G^{(\ell)}(x) + A_- F^{(\ell)}(x), & (9) \\ H^{(\ell+1)}(x) &= A_+ F^{(\ell)}(x) + A_- G^{(\ell)}(x) + b. & (10) \end{aligned}$$

The tropical forms of these equations using tropical arithmetic are given in [16].

From Proposition 2.1, we see that $F^{(\ell+1)}(x) = H^{(\ell+1)}(x) \oplus G^{(\ell+1)}(x)$ is the tropical sum of two tropical polynomials. Using this, we can determine the linear regions of a neural network by examining where $H^{(\ell+1)}$ meets $G^{(\ell+1)}$ at each layer. However, this is difficult to achieve with the current setup, because $H^{(\ell+1)}$ and $G^{(\ell+1)}$ depend directly on the network's input x. Therefore, we first need to decompose these recurrence relations to minimize their dependence on x.

We can decompose F, G, and H into the equations

$$\begin{aligned} F(x) &= F_A(x)x + f_b(x) & (11) \\ G(x) &= G_A(x)x + g_b(x) & (12) \\ H(x) &= H_A(x)x + h_b(x) & (13) \end{aligned}$$

where $F_A(x), G_A(x), H_A(x) \in \mathbb{R}^{n_{\ell+1} \times n_\ell}[x]$ are matrices, and where $f_b(x)$, $g_b(x)$, $h_b(x) \in \mathbb{R}^{n_{\ell+1}}$ are vectors.

Then the recurrence relations for F, G, and H given by (8) become:

$$\begin{aligned}
F_A^{(\ell+1)}(x) &= \mathbb{I}_{H>G} H_A^{(\ell+1)}(x) + \mathbb{I}_{H \leq G} G_A^{(\ell+1)}(x), \\
f_b^{(\ell+1)}(x) &= \mathbb{I}_{H>G} h_b^{(\ell+1)}(x) + \mathbb{I}_{H \leq G} g_b^{(\ell+1)}(x), \\
G_A^{(\ell+1)}(x) = A_+ G_A^{(\ell)}(x) + A_- F_A^{(\ell)}(x), &\quad H_A^{(\ell+1)}(x) = A_+ F_A^{(\ell)}(x) + A_- G_A^{(\ell)}(x), \\
g_b^{(\ell+1)}(x) = A_+ g_b^{(\ell)}(x) + A_- f_b^{(\ell)}(x), &\quad h_b^{(\ell+1)}(x) = A_+ f_b^{(\ell)}(x) + A_- g_b^{(\ell)}(x) + b
\end{aligned} \tag{14}$$

where $\mathbb{I}_{H>G}$ evaluates to 1 if $H^{(\ell+1)}(x) > G^{(\ell+1)}(x)$, and 0 otherwise.

This decomposition also requires us to have initial values for our recurrences. To find the initial values, consider an identity transformation $F^{(0)}(x) = I_{n \times n} \cdot x + \vec{0}$ (where n is the input size). This gives us that $F_A^{(0)}(x) = I_{n \times n}$ and $f_b^{(0)}(x) = \vec{0}$. We can also take $F_A^{(0)} = F_{A+}^{(0)} + F_{A-}^{(0)}$ to be the same type of decomposition into two non-negative matrices that we used when setting up (2). Then (2) tells us that $F_{A+}^{(0)} = H_A^{(0)}(x)$ and $F_{A-}^{(0)} = G_A^{(0)}(x)$, so our initial values are as follows:

$$F_A^{(0)}(x) = I_{n \times n} \qquad G_A^{(0)}(x) = 0_{n \times n} \qquad H_A^{(0)}(x) = I_{n \times n}, \tag{15}$$

$$f_b^{(0)}(x) = \vec{0} \qquad g_b^{(\ell+1)}(x) = \vec{0} \qquad h_b^{(\ell+1)}(x) = \vec{0}. \tag{16}$$

This recurrence relation gives way to the following result:

Theorem 2.2 *Let f be an L-layered neural network with ReLU activations and with n_1, n_2, ..., n_L neurons on each layer. Then any set of vectors $\{v_\ell \in \{-1, 1\}^{n_\ell} \mid \ell \in [L]\}$ uniquely specifies a single linear region of f, where the region is given by:*

$$\{x \in \mathbb{R}^n \mid \widetilde{H}(x, \ell) \leq \widetilde{G}(x, \ell) \text{ for all } \ell \in [L]\} \tag{17}$$

where

$$\widetilde{H}(x, \ell) := \texttt{diag}(v_\ell) \left(H_A^{(\ell)}(x) x + h_b^{(\ell)}(x) \right) \tag{18}$$

$$\widetilde{G}(x, \ell) := \texttt{diag}(v_\ell) \left(G_A^{(\ell)}(x) x + g_b^{(\ell)}(x) \right) \tag{19}$$

and where $\texttt{diag}(v_\ell) = \sum_{i=1}^{n_\ell} e_i^\intercal v_\ell e_i^\intercal$ is the $(n_\ell \times n_\ell)$-matrix where the (j, j)-th entry on the diagonal is given by $(v_i)_j$, and where all off-diagonal entries are zero.

Note that some of these regions may be empty. Also note that the set of vectors $v_i \mid_{i\in[L]}$ we construct is often referred to as an activation pattern.

Proof. We use induction. Let us take any set of vectors $v_i \mid_{i\in[L]}$ and construct the inequalities present in (17) one layer at a time, starting with the first layer, and examining the inequalities we introduce.

Base case.

The first layer introduces inequalities $\widetilde{H}(x,1) \leq \widetilde{G}(x,1)$. Using the initial values given by (15), we obtain:

$$\begin{aligned}
\widetilde{H}(x,1) &= \texttt{diag}(v_1)\Big(H_A^{(1)}(x)x + h_b^{(1)}(x)\Big) && (20)\\
&= \texttt{diag}(v_1)\Big(\left(A_+ I_{n_1\times n} + A_- 0_{n_1\times n}\right)x + \Big(A_+\vec{0} + A_-\vec{0} + b\Big)\Big) && (21)\\
&= \texttt{diag}(v_1)\big(A_+ x + b\big) && (22)
\end{aligned}$$

and

$$\begin{aligned}
\widetilde{G}(x,1) &= \texttt{diag}(v_1)\Big(G_A^{(1)}(x)x + g_b^{(1)}(x)\Big) && (23)\\
&= \texttt{diag}(v_1)\Big(\left(A_+ 0_{n_1\times n} + A_- I_{n_1\times n}\right)x + \Big(A_+\vec{0} + A_-\vec{0}\Big)\Big) && (24)\\
&= \texttt{diag}(v_1)A_- x && (25)
\end{aligned}$$

Hence the inequalities introduced are:

$$\begin{aligned}
\widetilde{H}(x,1) \leq \widetilde{G}(x,1) \quad &\iff \quad \texttt{diag}(v_1)\big(A_+ x + b\big) \leq \texttt{diag}(v_1)A_- x && (26)\\
&\iff \quad \texttt{diag}(v_1)\big(Ax + b\big) \leq 0 && (27)\\
&\iff \quad (v_1)_i\big(A_i x_i + b_i\big) \leq 0 \qquad \forall i \in [n_1] && (28)
\end{aligned}$$

where A_i is the i-th row of A. We have $(v_1)_i \in \{-1,+1\}$ for all i, so the inequalities in (28) tell us that the choice of -1 or $+1$ for each element of v_i is equivalent to choosing $A_i x_i + b_i \leq 0$ or $A_i x_i + b_i \geq 0$. Note that $A_i x_i + b_i$ is the pre-activation output of the i-th neuron of the first layer. By assumption, all activation functions are ReLU, which are linear on the region $x \leq 0$ and the region $x \geq 0$ (both of these inequalities are non-strict since ReLU is continuous).

Hence, each inequality of (28) (i.e. each row of the inequality $\widetilde{H}(x,1) \leq \widetilde{G}(x,1)$) limits the region (17) exclusively to either the positive half or the negative half of each neuron in the first layer. This completes the base case.

Inductive step.

Take some layer $\ell \in [L]$ of our network and suppose now that $\bigcap_{i=1}^{\ell}(\widetilde{H}(x,i) \leq \widetilde{G}(x,i))$ give the inequalities introduced by our choice of $v_i \mid_{i\in[L]}$ for layers 1 through ℓ. Then we have:

$$\begin{aligned}
\widetilde{H}(x,\ell+1) &= \texttt{diag}(v_{\ell+1})\Big(H_A^{(\ell+1)}(x)x + h_b^{(\ell+1)}(x)\Big) && (29)\\
&= \texttt{diag}(v_{\ell+1})\Big(\Big(A_+F_A^{(\ell)}(x) + A_-G_A^{(\ell)}(x)\Big)x \\
&\qquad + \Big(A_+f_b^{\ell}(x) + A_-g_b^{(\ell)}(x) + b\Big)\Big) && (30)
\end{aligned}$$

and

$$\begin{aligned}
\widetilde{G}(x,\ell+1) &= \texttt{diag}(v_{\ell+1})\Big(G_A^{(\ell+1)}(x)x + g_b^{(\ell+1)}(x)\Big) && (31)\\
&= \texttt{diag}(v_{\ell+1})\Big(\Big(A_+G_A^{(\ell)}(x) + A_-F_A^{(\ell)}(x)\Big)x \\
&\qquad + \Big(A_+g_b^{\ell}(x) + A_-f_b^{(\ell)}(x)\Big)\Big) && (32)
\end{aligned}$$

Subtracting $\widetilde{G}$ from $\widetilde{H}$, we get: $\widetilde{H}(x,\ell+1) - \widetilde{G}(x,\ell+1)$

$$\begin{aligned}
&= \texttt{diag}(v_{\ell+1})\Big(\Big((A_+ - A_-)F_A^{(\ell)}(x) - (A_+ - A_-)G_A^{(\ell)}(x)\Big)x \\
&\qquad + \Big((A_+ - A_-)f_b^{\ell}(x) - (A_+ - A_-)g_b^{(\ell)}(x) + b\Big)\Big) && (33)\\
&= \texttt{diag}(v_{\ell+1})\Big(A\Big(F^{(\ell)}(x) - G^{(\ell)}(x)\Big) + b\Big) && (34)\\
&= \texttt{diag}(v_{\ell+1})\Big(A\nu^{(\ell)}(x) + b\Big) && (35)
\end{aligned}$$

where $\nu^{(\ell)}(x)$ is the post-activation output of the previous layer ℓ, as specified in Proposition 2.1. Hence, the inequality $\widetilde{H}(x,\ell+1) \leq \widetilde{G}(x,\ell+1)$ is equivalent to:

$$\begin{aligned}
&\widetilde{H}(x,\ell+1) - \widetilde{G}(x,\ell+1) \leq 0 \\
&\quad\iff \quad \texttt{diag}(v_{\ell+1})\Big(A\nu^{(\ell)}(x) + b\Big) \leq 0 && (36)\\
&\quad\iff \quad (v_{\ell+1})_i\big(A_i(\nu^{(\ell)}(x))_i + b_i\big) \leq 0, \qquad \forall i \in [n_{\ell+1}]. && (37)
\end{aligned}$$

Using the same reasoning as our base case, we see that the choice of each entry of $v_{\ell+1}$ corresponds to choosing whether we take $A_i(\nu^{(\ell)}(x))_i + b_i \leq 0$ or $A_i(\nu^{(\ell)}(x))_i + b_i \geq 0$ for each neuron. $A_i(\nu^{(\ell)}(x))_i + b_i$ is the pre-activation output of the $(\ell+1)$-th layer, so this choice is equivalent to choosing the positive or negative side of the ReLU activation of each neuron in the $(\ell+1)$-th layer.

Conclusion of proof.

Setting $\ell = L-1$, we obtain that the inequalities $\bigcap_{i=1}^{L}(\widetilde{H}(x,i) \leq \widetilde{G}(x,i))$ correspond exactly to a single linear region of the network's final output layer, giving (17), which completes the proof.

Q.E.D.

3 An Algorithm to Find Linear Regions

Theorem 2.2 now facilitates Algorithm 1 for finding the linear region specified by any set of vectors $v_\ell \mid_{l\in[L]}$, without the need to provide a direct input x to the neural network. A visual representation of Algorithm 1 is given in Fig. 1.

The entries of vectors $v_\ell \mid_{\ell\in[L]}$ uniquely correspond to a single linear region, which may be empty or nonempty. This means that Algorithm 1 allows us to find *all* linear regions. A brute-force method for doing this would be to enumerate all possible assignments of -1 and $+1$ to entries in v_ℓ, which would have exponential complexity in the total number of neurons, since the number of assignments is $2^{\sum_{\ell=1}^{L} n_\ell}$. Instead, we can reduce the complexity by using a tree traversal approach, which is laid out in Algorithm 2.

Let us now follow a concrete walkthrough of Algorithm 1 on a small example network. For this example, consider a network with a 2-dimensional input layer, 2 hidden layers with 2 neurons per layer, and a 1-dimensional output layer. Suppose that the weights and biases of each hidden layer in the network are:

$$A^{(1)} = \begin{bmatrix} -4 & 1 \\ -4 & -1 \end{bmatrix}, \quad b^{(1)} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \quad A^{(2)} = \begin{bmatrix} -8 & 3 \\ -\frac{21}{4} & \frac{19}{4} \end{bmatrix}, \quad b^{(2)} = \begin{bmatrix} -4 \\ 1 \end{bmatrix}. \tag{38}$$

Let us assume that the 1-dimensional output layer does not have an activation function, so we can ignore it and instead focus on the two hidden

layers.

Take the arbitrary truth assignment $v_1 = \begin{bmatrix}1\\1\end{bmatrix}$, $v_2 = \begin{bmatrix}1\\-1\end{bmatrix}$. Each entry corresponds to whether choosing either the positive or negative half of the activation function for each neuron in the network, so we want to find the linear region that corresponds to this truth assignment.

Using Algorithm 1, on line 2, we first set the initial values $F_A^{(0)}$, $G_A^{(0)}$, $H_A^{(0)}$, $f_b^{(0)}$, $g_b^{(0)}$, $h_b^{(0)}$ to the initial values given by (15). Then, on line 7 of the algorithm, we apply the recurrence relations given by (14), giving the following:

$$G_A^{(1)} = \begin{bmatrix} 4 & 0 \\ 4 & 1 \end{bmatrix}, \quad g_b^{(1)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad H_A^{(1)} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad h_b^{(1)} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}. \tag{39}$$

Next, lines 8-20 of Algorithm 1 allow us to construct $F_A^{(1)}$ and $f_b^{(1)}$ by using v_1 to select from $G_A^{(1)}$ and $g_b^{(1)}$, or from $H_A^{(1)}$ and $h_b^{(1)}$ for each row. Since we have $v_1 = \begin{bmatrix}1\\1\end{bmatrix}$, we select $G_A^{(1)}$ and $g_b^{(1)}$ for both rows, giving:

$$F_A^{(1)} = \begin{bmatrix} 4 & 0 \\ 4 & 1 \end{bmatrix}, \quad f_b^{(1)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{40}$$

The entries of v_1 correspond to selecting a half space for each neuron on the first layer, which produce a linear region when all such half spaces are intersected. These half spaces are specified by the inequalities on lines 14 and 18 of Algorithm 1. Hence, at the end of the first layer, the linear region is given by:

$$\left(\begin{bmatrix} 4 & 0 \\ 4 & 1 \end{bmatrix} - \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \right) \begin{bmatrix} x \\ y \end{bmatrix} + \left(\begin{bmatrix} 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 2 \\ 3 \end{bmatrix} \right) \leq \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{41}$$

Next, we must apply the same recurrence relations to the second layer, giving:

$$G_A^{(2)} = \begin{bmatrix} 12 & 11 \\ 19 & 10 \end{bmatrix}, \quad g_b^{(2)} = \begin{bmatrix} 16 \\ \frac{21}{2} \end{bmatrix}, \quad H_A^{(2)} = \begin{bmatrix} 32 & 0 \\ 21 & 0 \end{bmatrix}, \quad h_b^{(2)} = \begin{bmatrix} 12 \\ \frac{23}{2} \end{bmatrix}. \tag{42}$$

For layer 2, our truth assignment is $\begin{bmatrix}1\\-1\end{bmatrix}$, so we select $F_A^{(2)}$ and $f_b^{(2)}$ by taking a mixture of both $G_A^{(2)}$ & $g^{(2)}$, and $H_A^{(2)}$ & $h_b^{(2)}$. This gives us:

$$F_A^{(2)} = \begin{bmatrix} 32 & 0 \\ 19 & 10 \end{bmatrix}, \quad f_b^{(2)} = \begin{bmatrix} 12 \\ \frac{21}{2} \end{bmatrix}. \tag{43}$$

Finally, we obtain the bounds that this layer introduces to our linear region by again taking the inequalities on lines 14 and 18 of Algorithm 1, giving:

$$\left(\begin{bmatrix} 12 & 11 \\ 21 & 0 \end{bmatrix} - \begin{bmatrix} 32 & 0 \\ 19 & 10 \end{bmatrix} \right) \begin{bmatrix} x \\ y \end{bmatrix} + \left(\begin{bmatrix} 16 \\ \frac{23}{2} \end{bmatrix} - \begin{bmatrix} 12 \\ \frac{21}{2} \end{bmatrix} \right) \leq \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{44}$$

Note that the top rows of all matrix subtractions correspond to taking $G_A^{(2)} - H_A^{(2)}$, while the bottom rows correspond to $H_A^{(2)} - G_A^{(2)}$. The inequalities given by (44) can be expressed as $\{(x, y) \in \mathbb{R}^2 \mid -20x + 11y \leq -4, 2x - 10y \leq -1\}$.

Taking the intersection of the half spaces given by (41) and (44), we obtain our final linear region, which is given by:

$$\left\{ (x, y) \in \mathbb{R}^2 \;\middle|\; \begin{array}{rcl@{\qquad}rcl} 4x - y & \leq & 2, & 4x + y & \leq & 3 \\ -20x + 11y & \leq & -4, & x - 10y & \leq & -1 \end{array} \right\}. \tag{45}$$

Fig. 1 shows a visualization of this example, where the constraints imposed on the linear region by each neuron are shaded in blue for the region $[0, 1] \times [0, 1] \subset \mathbb{R}^2$. The resulting linear region shown on the right side of the figure exactly matches the constraints given by (45).

If we were to repeat this process for all possible truth assignments v_1 and v_2, we would obtain all linear regions of the network. Algorithm 2 allows us to efficiently explore all such truth assignments.

4 Linear Regions in Networks With Skip Connections

Using the results of Sect. 2, we can also find linear regions of networks with skip connections — we need only update our recurrence relations. In Sect. 2, our recurrences relied on the fact that the post-activation output ν of the $(\ell + 1)$-th layer was given by $\nu^{(\ell+1)}(x) = \max\{A\nu^{(\ell)}(x) + b, 0\}$. To account for skip connections, suppose that for any layer $\ell + 1$, there is some subset $K \subset [\ell]$ of layers preceding layer $\ell + 1$ that have a skip connection leading to layer $\ell + 1$. Then the input to layer $\ell + 1$ is $\nu^{(\ell)}(x) + \sum_{k \in K} \nu^{(k)}(x)$. Therefore, the recurrence relation for the post-activation output of layer $\ell + 1$ is:

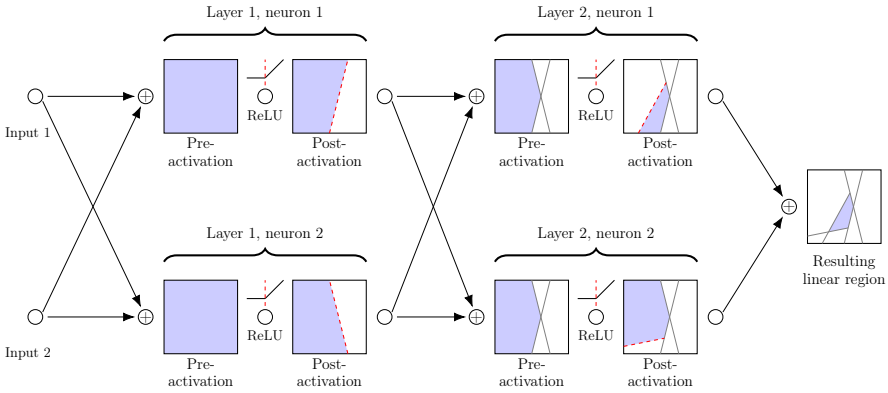

Figure 1: A visualization of the walkthrough of Algorithm 1 given in Sect. 3. For each neuron with ReLU activation, we split the input space in two by introducing a hyperplane and by selecting a half space with $(v_\ell)_i$. The newly-introduced hyperplanes at each neuron (as given by Theorem 2.2) are highlighted in red, and the shaded blue area represents the selected intersection of half spaces that comprise the linear region.

$$\begin{aligned}
\nu^{(\ell+1)}(x) &= \max\left\{A\Big(\nu^{(\ell)}(x) + \sum_{k\in K}\nu^{(k)}(x)\Big) + b, 0\right\} && (46)\\
&= \max\Big\{A_+\nu^{(\ell)}(x) + b + A_+\sum_{k\in K}\nu^{(k)}(x), A_-\nu^{(l)}(x)\\
&\qquad + A_-\sum_{k\in K}\nu^{(k)}(x)\Big\} - \Big(A_-\nu^{(l)}(x) + A_-\sum_{k\in K}\nu^{(k)}(x)\Big) && (47)
\end{aligned}$$

We can express $\nu^{(k)}(x)$ as $(F^{(k)}(x) - G^{(k)}(x))$, giving:

$$\begin{aligned}
F^{(\ell+1)}(x) &= \max\{H^{(\ell+1)}(x), G^{(\ell+1)}(x)\} && (48)\\
G^{(\ell+1)}(x) &= A_+G^{(\ell)}(x) + A_-F^{(\ell)}(x) + \underline{A_-\sum_{k\in K}\Big(F^{(k)}(x) - G^{(k)}(x)\Big)} && (49)\\
H^{(\ell+1)}(x) &= A_+F^{(\ell)}(x) + A_-G^{(\ell)}(x) + \underline{A_+\sum_{k\in K}\Big(F^{(k)}(x) - G^{(k)}(x)\Big)} && (50)
\end{aligned}$$

Here, the altered parts are underlined for emphasis. Lastly, we need recurrence relations for H_A, h_b, G_A, and g_b, since this allows us to calculate the

boundaries of linear regions. These relations are as follows:

$$G_A^{(\ell+1)}(x) = A_+ G_A^{(\ell)}(x) + A_- F_A^{(\ell)}(x) + \underline{A_- \sum_{k\in K} \Big(F_A^{(k)}(x) - G_A^{(k)}(x) \Big)} \quad (51)$$

$$g_b^{(\ell+1)}(x) = A_+ g_b^{(\ell)}(x) + A_- f_b^{(\ell)}(x) + \underline{A_- \sum_{k\in K} \Big(f_b^{(k)}(x) - g_b^{(k)}(x) \Big)} \quad (52)$$

$$H_A^{(\ell+1)}(x) = A_+ F_A^{(\ell)}(x) + A_- G_A^{(\ell)}(x) + \underline{A_+ \sum_{k\in K} \Big(F_A^{(k)}(x) - G_A^{(k)}(x) \Big)} \quad (53)$$

$$h_b^{(\ell+1)}(x) = A_+ f_b^{(\ell)}(x) + A_- g_b^{(\ell)}(x) + \underline{A_+ \sum_{k\in K} \Big(f_b^{(k)}(x) - g_b^{(k)}(x) \Big)} + b \quad (54)$$

and F_A and f_b are unchanged from before. The conclusion of this section is that the relations given in (51) can be used in Algorithms 1 and 2 to find all linear regions of neural networks with arbitrarily many skip connections.

5 Visualization of Linear Regions in Neural Networks

In this section, we use Algorithms 1 and 2 to plot *all* linear regions in some small neural networks, and we analyze their appearance.

For easy visualization, we consider regression problems on neural networks with 2 scalar input variables x and y, and 1 scalar output variable z. We perform experiments with each of the following functions:

$$\phi_1(x,y) = \sin(\log|x| + \log|y|) \quad (55)$$

$$\phi_2(x,y) = \sin(\sqrt{x^2+y^2}) \div \sqrt{x^2+y^2} \quad (56)$$

$$\phi_3(x,y) = \sin\big(\cos(x/2)\big) \sin\big(\cos(y/2)\big) \quad (57)$$

$$\phi_4(x,y) = \sin\big(\tan(x/2)\big) \sin\big(\tan(y/2)\big) \quad (58)$$

ϕ_1, ϕ_2, and ϕ_3 are examples of functions that a neural network can be expected to model well. On the other hand, ϕ_4 is given as an unreasonably difficult function for a small network to model, which we expect to lead to overfitting, allowing us to observe the linear regions in such a scenario. These functions are plotted in the left column of Fig. 2. For each of these ϕ functions, we train simple neural networks with the Adam optimizer and with learning rate $\eta = 0.01$ for 20,000 epochs. Each training instance (x, y) is sampled from $[-10, 10] \times [-10, 10]$ on a uniform grid at intervals of 0.1.

For each function, two networks are trained. Both have ReLU activation on all hidden nodes, and both have 5 neurons per layer The only difference between the two networks is that one has 2 hidden layers, while the other has 5 hidden layers, allowing us to observe the effects of adding more layers. This is not the first experiment to produce visualizations of linear regions, as there is precedence from Montúfar et al. [12], Hanin and Rolnick [4], and X. Zhang and Wu [17], who have all made similar visualizations. However, we believe that this is the first experiment that captures *all* distinct linear regions of a neural network and provides exact boundaries for these regions.

The resulting linear regions of the trained networks are shown in Fig. 2. From this, we make the following observations:

- For ϕ_1 and ϕ_2, the 5-layer network partitioned the input region into regions that mimic the patterns in the training data. The 2-layer network also produced visual patterns matching the training data in less detail, suggesting that deeper networks may be able to position boundaries of linear regions more effectively than shallow networks, which partially explains their increased effectiveness (among many other known explanations).

- ϕ_3 had contours that were more difficult to model than ϕ_1 and ϕ_2, but the patterns produced by the 5-layer network are still sensible. The deeper nature of the 5-layer network over the 2-layer network was necessary to capture this pattern, where early layers produced long, sweeping hyperplanes surrounding each peak and valley, and successive layers captured the smaller radial patterns. Our network only had 5 neurons per layer, yet 9 such patterns were made, suggesting that the divisions produced in early layers allow the input space to be "folded" in such a way that patterns produced by later layers are repeated across the space. This folding analogy is detailed by Montúfar et al. [12].

- On the other hand, for ϕ_4 (which we expect to induce overfitting), the networks produced clusters of tiny linear regions near the center, with highly non-uniform regions near the edges. This shows us that a neural network that overfits on a task may start dividing the input space randomly via early layers, before subdividing resulting small regions further. This is consistent with an intuitive understanding of overfitting, which is that it occurs when too much emphasis is placed on noise in training data.

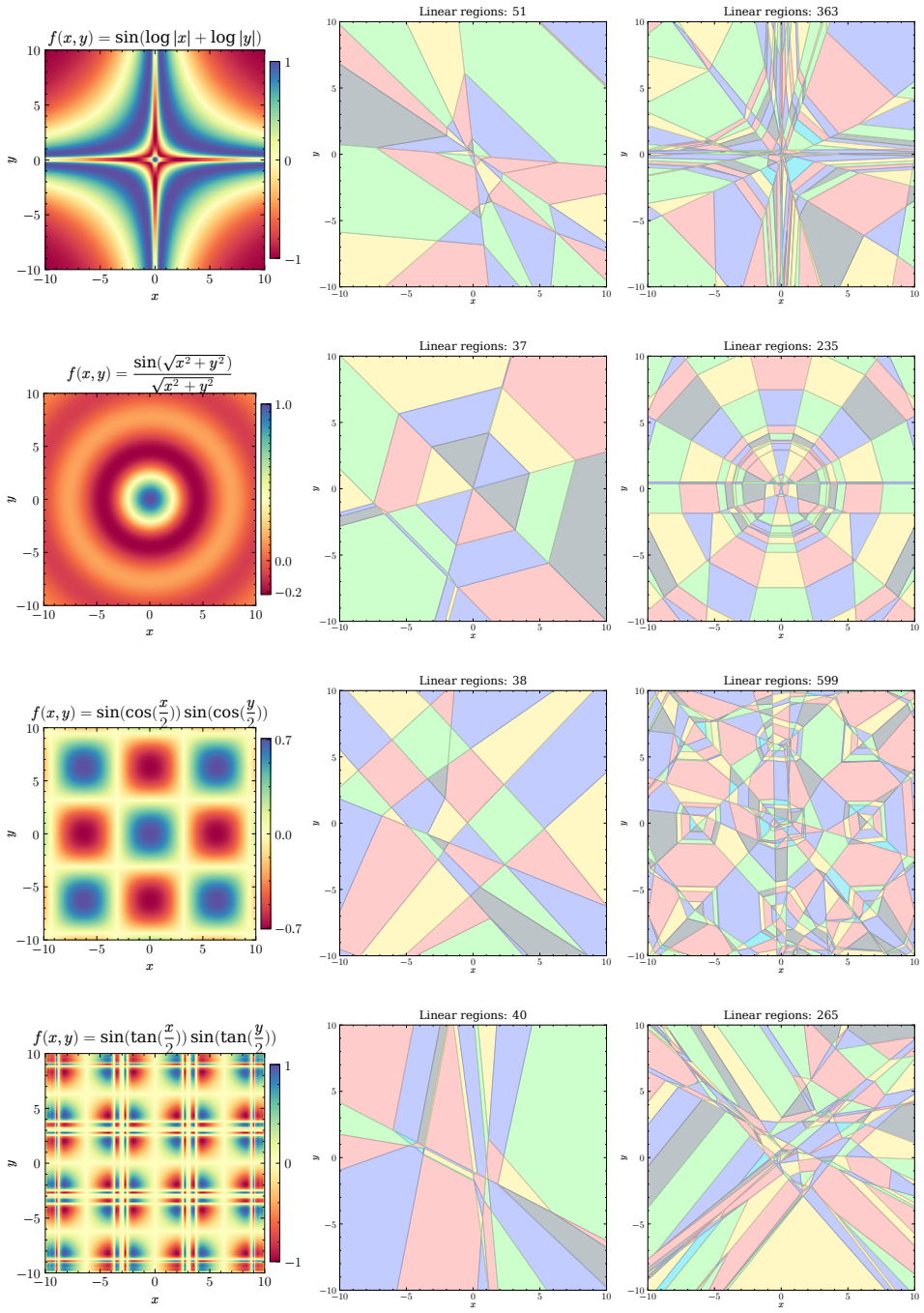

Figure 2: Linear regions of neural networks after 20,000 training epochs. Each row i corresponds to a function ϕ_i in (55). Colors are arbitrary. *Left column:* Plot of function given as training data. *Middle column:* Linear regions of 2-layer neural network. *Right column:* Linear regions of 5-layer neural network.

6 Comparison With and Without Skip Connections

In this section, we compare the number of linear regions in neural networks with and without skip connections. We start in Subsect. 6.1 by making this comparison throughout training, where we find that neural networks with skip connections have more linear regions than regular networks, even with no training. Given this unexpected result, we verify it with hypothesis tests in Subsect. 6.2.

6.1 Comparison Throughout Training

We train two types of neural networks on each of the functions ϕ_1, ϕ_2, ϕ_3, ϕ_4 given in (55-58). We create the training data in the same way as in Sect. 5, though we also add a testing set, where we sample at the same interval as the training set, and offset the samples by 0.1 (e.g. the first few x coordinates in the training set are -10, -9.8, -9.6, ..., while the first few x coordinates of the testing set are -9.9, -9.7, -9.5, ...). Both types of networks have 5 hidden layers, with 4 neurons per layer. One type of network does not have skip connections, while the other has a skip connection from the first layer to the third layer, and from the second layer to the fourth layer. For each type of network, we perform 10 trials on each function ϕ_1 through ϕ_4, and take the overall mean across all trials and functions of the number of linear regions throughout training. We also record the loss to validate our methodology.

The results are shown in Fig. 3. We observe that the networks with skip connections consistently have more linear regions than the networks without skip connections throughout all stages of training. This result is surprising, as the number of neurons is the same in both types of network, and so the upper bound on the number of linear regions is the same. Ostensibly, one might expect there to be no difference between the two networks in the early stages of training, but one may expect a difference to appear after training. Given the nature of this result, we perform additional hypothesis testing in Subsect. 6.2 to verify this result for randomly-initialized networks.

6.2 Comparison Upon Random Initialization

In Subsect. 6.1, we found evidence that networks with skip connections may have more linear regions than equivalent networks without skip connections, even without training. In this subsection, we perform hypothesis tests to verify this result.

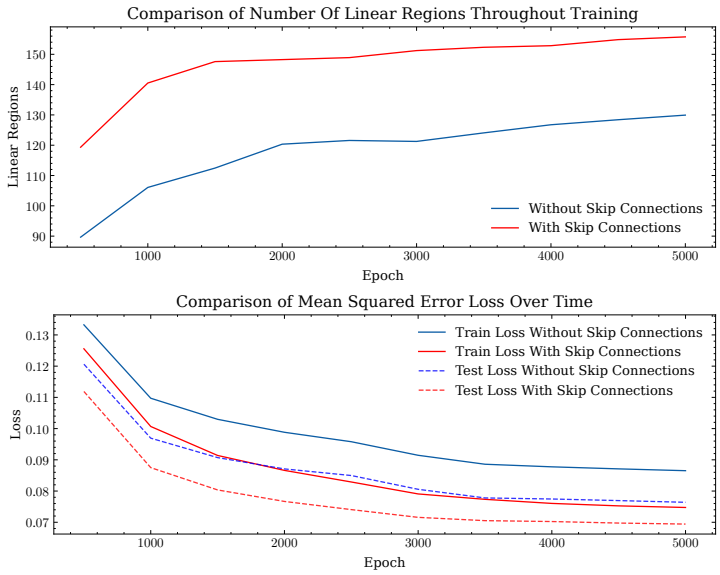

Figure 3: Comparison of neural networks with and without skip connections throughout training. Networks were trained on functions ϕ_1, ϕ_2, ϕ_3, ϕ_4, and the mean was taken over 10 trials of each. *Top:* Number of linear regions. *Bottom:* Mean squared error loss.

For each test, we randomly initialize 50 neural networks without skip connections using Kaiming He initialization, each with some given number of layers and neurons per layer (as laid out in the *"neural network parameters"* columns of Table 1), and we initialize 50 networks with skip connections. Taking μ_1 to be the sample mean number of linear regions in the networks without skip connections and μ_2 to be the sample mean with skip connections, we apply a one-tailed Mann-Whitney U test with null hypothesis $\mu_1 = \mu_2$ and alternative hypothesis $\mu_1 < \mu_2$, with significance level 0.05. We use a U test because the distribution of each sample is not Gaussian. Further, we find that the best-fitting distribution for each sample is a shifted Gamma distribution. Therefore, we also include goodness-of-fit tests for Gamma distributions for each sample, and we find no significant evidence to suggest that this Gamma distribution is inappropriate.

The results are given in Table 1. In each test, we obtain small p-values and reject the null hypothesis. These tests provide us with strong evidence that the average number of linear regions for untrained networks with skip connections is greater than that of equivalent untrained networks without skip connections.

Table 1: Results of hypothesis tests for the sample mean number of linear regions of neural networks with and without skip connections. For each row, 50 networks with skip connections and 50 networks without skip connections are initialized. An entry of e.g. $1 \to 3$ in the *"Skip connections"* column denotes that the output of the first layer is added to the input of the third layer. *"KS test"* is a one-sample Kolmogorov-Smirnov test for the goodness-of-fit of a Gamma distribution. One-tailed U-test compares the sample mean number of linear regions for networks with and without skip connections.

Neural network parameters			KS test p-value		Sample mean no. of linear regions		One-tailed Mann-Whitney U test		
No. of layers	Neurons per layer	Skip connections	With skips	Without skips	With skips	Without skips	U-statistic	U test p-value	Reject null hypothesis
6	5	$1 \to 3$, $2 \to 4$, $3 \to 5$	0.960	0.830	206.8	117.7	2237.5	1.0×10^{-11}	Yes
6	4	$1 \to 3$, $2 \to 4$, $3 \to 5$	0.925	0.737	111.6	62.2	2161.5	3.4×10^{-10}	Yes
5	4	$1 \to 3$, $2 \to 5$	0.801	0.709	96.7	53.3	2242.0	8.1×10^{-12}	Yes
4	4	$1 \to 3$, $2 \to 5$	0.849	0.984	71.5	53.5	1804.5	1.3×10^{-4}	Yes
3	6	$1 \to 3$	0.815	0.665	137.6	115.7	1711.5	1.5×10^{-3}	Yes
3	4	$1 \to 3$	0.810	0.992	52.3	42.4	1728.0	9.9×10^{-4}	Yes

6.3 Analysis of Results

In Subsect. 6.1, we observed that neural networks with skip connections have more linear regions throughout training than regular networks, and achieve a greater maximum number of skip connections when training finishes. This indicates that networks with skip connections produce a more advanced output map with more facets, and are therefore more expressive than regular networks.

One explanation of this observation is that layers directly following skip connections have more flexibility to partition either the space from the previous layer, or partition the space from the "skipped" layers. More precisely, suppose we have a skip connection from layer k to layer $\ell + 1$, so that the input to the $(\ell + 1)$-th layer is $\nu^{(k)}(x) + \nu^{(\ell)}(x)$. Then layer $(\ell + 1)$ introduces hyperplanes with equations $G^{(\ell+1)}(x) = H^{(\ell+1)}(x)$, and by (48), this is equivalent to:

$$G^{(\ell)}(x) + A_{-}\nu^{(k)}(x) = H^{(\ell)}(x) + A_{+}\nu^{(k)}(x) \tag{59}$$

On the other hand, a neural network without this skip connection would

introduce hyperplanes at $G^{(\ell)}(x) = H^{(\ell)}(x)$. Therefore, when training a neural network, each skip connection provides the optimization algorithm with an opportunity to introduce a hyperplane as a function of the outputs of both layers ℓ and k, rather than layer ℓ alone, leading to more linear regions.

In Subsect. 6.2, we found evidence that the average number of linear regions for untrained neural networks with skip connections is greater than that of equivalent networks without skip connections. This shows that adding skip connections makes a network inherently more expressive, rather than simply improving its ability to be optimized with gradient descent methods. To reason this out, consider that a layer that uses a skip connection as its input introduces a hyperplane at (59) — the matrices A_+ and A_- have random entries, and so a random multiple of $\nu^{(k)}$ is added to the hyperplane equation. This aligns the direction of the new hyperplane more with $\nu^{(k)}$, making it more likely to intersect each existing linear region and to split them into two. This result may be possible to formally prove by using random variables to represent the coefficients of the boundaries of linear regions, which we foresee as a future research direction.

7 Application: Caching Values in Neural Networks

In this section, we propose a method to compress nonlinear neural networks into as a single layer (which we originally envisioned in [7]), given by Algorithm 3.

As a caching algorithm, the speedup we observe depends entirely on how often each new instance arrives pre-cached, which is highly dependent on both the complexity of the neural network, as well as the data set on which the network has been trained. This raises the question of how often trained models cache instances, and whether networks with skip connections cache data at a different rate than networks without skip connections. In the remainder of this section, we seek answers to these questions by performing experiments with our caching algorithm. In Subsect. 7.1, we train classifier models of various sizes on three different data sets. In Subsect. 7.2, we examine the effect of using skip connections on the rate at which data instances are cached.

7.1 Caching Experiment on Various Data Sets

To examine how often cached results are used, we train models of varying sizes on the following three different data sets: *a*) MNIST data set of hand-

written digits, containing 60,000 training instances and 10,000 test instances, *b*) CIFAR-10 data set [9], containing 50,000 training instances and 10,000 testing instances (for computational tractability, we desaturate the 3-channel RGB images into 1-channel grayscale images), *c*) Street View House Number (SVHN) data set [13], containing 73,257 training instances and 26,032 testing instances.

For each of the three data sets, we train a network with 4 layers and 8 neurons per layer, a network with 3 layers and 16 neurons per layer, and a network with 3 layers and 32 neurons per layer. Each network is trained on each data set with the Adam optimizer for 5 epochs with learning rate $\eta = 5 \times 10^{-4}$.

After training, we cache all linear regions that contain at least one training instance by storing a string with a 1 or 0 for each neuron, indicating whether the neuron's pre-activation function is positive or negative for the given training instance. For each testing instance, we then find the linear region in which the instance lies, and check whether any training instance lies in the same region.

The results are shown in Fig. 4. For easier visualization, we use Uniform Manifold Approximation and Projection (UMAP) [11] to plot the data in 2D. Instances in the test set that lie in the same linear region as at least one training instance are colored in blue, while the rest of the test set is in red. For simplicity, let us refer to these types of instances as cached and uncached, respectively.

Based on Fig. 4, we can make several observations. *a*) Rates of caching are not uniform across each class. For instance, in the plot corresponding to the 3-layer, 32-neuron-per-layer model on MNIST (top-right position of Fig. 4), we observe a mix of cached and uncached test instances, and we see that the labels "9" and "1" have high rates of caching, while "2" and "3" are much lower. *b*) From the middle column of Fig. 4, we see that uncached instances tend to appear further away from class centroids, while the cached instances are usually closer to the centroids. This shows that there may be potential for this method to be able to detect out-of-distribution data. Such a method could involve mapping out linear regions as a graph, and counting the path length to the nearest in-distribution element. *c*) The proportion of cached elements depends on the number of neurons in the network. For larger networks (right column of Fig. 4), we have more linear regions, so it is less likely that any two data points lie in the same region. To use this technique on large networks, knowledge distillation techniques may be needed. *d*) The data set choice affects the rate of caching. This is most evident when comparing CIFAR-10 and MNIST, which have the same

number of testing instances, but more test data is cached on MNIST than for CIFAR-10, which may be due to the higher relative difficulty of CIFAR-10.

7.2 Caching Experiment With and Without Skip Connections

Using a similar setup as in Sect. 7.1, we can also compare networks with and without skip connections. For each model-dataset pair as given in Fig. 4, we conduct a trial where we train 50 networks with skip connections, and 50 networks without skip connections, and record the number of cached values. For each trial, take μ_1 to be the sample mean number of cached instances of the networks without skip connection and μ_2 to be the sample mean of the networks with skip connections — we apply a one-tailed Mann-Whitney U test with null hypothesis $\mu_1 = \mu_2$ and alternative hypothesis $\mu_1 > \mu_2$, with significance level 0.05 for each test. The results of these hypothesis tests are given in Table 2.

Table 2: Results of hypothesis tests for the sample mean number of cached values found in the testing set for various data sets. For each row, 50 networks with skip connections (as in *"Skip connections"* column, which uses the same format as Table 1) are trained, and 50 networks without skip connections are trained. All networks are trained for 5 epochs on MNIST, CIFAR-10, or SVHN, respectively. Using Algorithm 3, the number of cached test instances is recorded for each trained model. A one-tailed Mann-Whitney U test is applied to compare the means with and without skip connections.

Data set	Neural network parameters			Mean accuracy on test set		Mean no. of cached values in test set		One-tailed Mann-Whitney U test		
	No. of layers	Neurons per layer	Skip connections	With skips	Without skips	With skips	Without skips	U-statistic	U test p-value	Reject null hypothesis
MNIST	4	8	1→3, 2→4	83.8%	78.7%	9966.4	9970.0	1245.5	0.489	No
	3	16	1→3	91.8%	91.3%	9211.6	9349.2	997.5	0.041	Yes
	3	32	1→3	94.4%	93.9%	2140.3	3200.6	397.5	2.1×10^{-9}	Yes
CIFAR-10	4	8	1→3, 2→4	30.3%	29.2%	8965.6	9125.4	850.5	3.0×10^{-3}	Yes
	3	16	1→3	35.8%	35.2%	1702.2	2436.5	314.0	5.6×10^{-11}	Yes
	3	32	1→3	39.0%	39.0%	4.1	7.3	485.0	5.7×10^{-8}	Yes
SVHN	4	8	1→3, 2→4	52.5%	48.8%	24806.9	24982.8	877.0	5.1×10^{-3}	Yes
	3	16	1→3	64.9%	63.4%	10273.2	12532.9	304.5	3.6×10^{-11}	Yes
	3	32	1→3	72.6%	71.6%	87.3	174.7	289.5	1.8×10^{-11}	Yes

From Table 2, we reject the null hypothesis for all but one model-dataset pair[1]. We also observe that the networks with skip connections have greater or equal average accuracy than the networks without skip connections, which aligns with conventional knowledge. When considered together with the fact that this accuracy comes from uncached data, this indicates that introducing skip connections to a neural network allows it to generalize to unseen data more effectively. Combined with our observation in Sect. 6 that neural networks with skip connections have more linear regions, this shows that networks with skip connections do not simply introduce linear regions around individual training instances, but instead produce regions that are representative of underlying patterns.

8 Conclusion

Recurrence relations form the basis for algorithms to compute all linear regions of a neural network, generalized to work for networks with skip connections. Visualizations of those linear regions help to understand how patterns in the training data may lead to overfitting. Caching the linear transformations allows for faster predictions. Through experimentation, we found that skip connections allow a model to both be more expressive by creating more advanced output maps, while also ensuring that these maps can generalize to new data.

References

[1] R. Balestriero and R. G. Baraniuk. Mad Max: Affine spline insights into deep learning. *Proceedings of the IEEE*, 109(5):704–727, 2021.

[2] R. Balestriero, A. I. Humayun, and R. G. Baraniuk. On the geometry of deep learning. *Notices of the American Mathematical Society*, 72(4):374–385, 2025.

[3] H. Edelsbrunner. *Algorithms in combinatorial geometry*. Springer, 1987.

[4] B. Hanin and D. Rolnick. Complexity of linear regions in deep networks. In *Proceedings of the 36th International Conference on Machine*

[1] We did not reject the null hypothesis when using the 4-layer network on MNIST. Here, almost all test instances were cached regardless of the presence of skip connections ($\frac{9966.4}{10000} \approx 99.7\%$ vs $\frac{9970}{10000} \approx 99.7\%$), giving nearly equal population means.

Learning, volume 97 of *Proceedings of Machine Learning Research*, pages 2596–2604. PMLR, 2019.

[5] A. I. Humayun, R. Balestriero, and R. Baraniuk. Training dynamics of deep network linear regions. `https://arxiv.org/abs/2310.12977`, 2023.

[6] J. Joyce. *Algebraic Frameworks and Computational Experiments for Enhanced Machine Learning*. PhD thesis, University of Illinois at Chicago, 2025.

[7] J. Joyce and J. Verschelde. Faster predictions in convolutional neural networks. In F. Boulier, C. Mou, T.M. Sadykov, and E.V. Vorozhtsov, editors, *Proceedings of the 26th International Workshop on Computer Algebra in Scientific Computing (CASC)*, volume 14938 of *Lecture Notes in Computer Science*, pages 161–177. Springer-Verlag, 2024.

[8] Johnny Joyce. GitHub repository. `https://github.com/johnnyvjoyce/tropical-geometry-linear-regions`, 2025.

[9] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, Toronto, Canada, 2009.

[10] P. Maragos, V. Charisopoulos, and E. Theodosis. Tropical geometry and machine learning. *Proceedings of the IEEE*, 109(5):728-755, 2021.

[11] L. McInnes, J. Healy, N. Saul, and L. Großberger. UMAP: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29), 2018.

[12] G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. *NeurIPS*, 27, 2014.

[13] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NeurIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 4, 2011.

[14] T. Serra, C. Tjandraatmadja, and S. Ramalingam. Bounding and counting linear regions of deep neural networks. In *International conference on machine learning*, pages 4558-4566. PMLR, 2018.

[15] C. D. Tóth. Binary space partitions: recent developments. *Combinatorial and Computational Geometry*, 52:525-552, 2005.

[16] L. Zhang, G. Naitzat, and L.-H. Lim. Tropical geometry of deep neural networks. In *International Conference on Machine Learning*, pages 5824-5832. PMLR, 2018.

[17] X. Zhang and D. Wu. Empirical studies on the properties of linear regions in deep neural networks. In *International Conference on Learning Representations*, 2020.

Algorithm 1 Method to find linear regions of neural networks, as in Theorem 2.2.

function FindLinearRegion($v_1, \ldots, v_L$)
  $F_A^{(0)}, G_A^{(0)}, H_A^{(0)}, f_b^{(0)}, g_b^{(0)}, h_b^{(0)} \leftarrow$ Initial values laid out in (15)
  $S \leftarrow$ Empty array ▷ *To be filled with inequalities that define the linear region.*
  for $l = 0, \ldots, L-1$ **do**
    $A \leftarrow$ weight matrix of layer $\ell + 1$
    $b \leftarrow$ bias vector of layer $\ell + 1$
    $G_A^{(l+1)}, H_A^{(l+1)}, g_b^{(\ell+1)}, h_b^{(\ell+1)} \leftarrow$ Values for the next layer laid out in (14)
    $F_A^{(\ell+1)} \leftarrow$ a blank matrix to be filled out
    $f_b^{(\ell+1)} \leftarrow$ a blank vector to be filled out
    for $i = 1, \ldots, l$ **do**
      $\vec{\alpha} \leftarrow i$-th row of $H_A^{(\ell+1)} - G_A^{(\ell+1)}$
      $\beta \leftarrow i$-th entry of $h_b^{(\ell+1)} - g_b^{(\ell+1)}$
      if $(v_{l+1})_i == +1$ **then**
        Append the inequality $\vec{\alpha} \cdot x + \beta \leq 0$ to S
        i-th row of $F_A^{(\ell+1)} \leftarrow i$-th row of $G_A^{(\ell+1)}$
        i-th entry of $f_b^{(\ell+1)} \leftarrow i$-th entry of $g_b^{(\ell+1)}$
      else
        Append the inequality $-\vec{\alpha} \cdot x - \beta \leq 0$ to S
        i-th row of $F_A^{(\ell+1)} \leftarrow i$-th row of $H_A^{(\ell+1)}$
        i-th entry of $f_b^{(\ell+1)} \leftarrow i$-th entry of $h_b^{(\ell+1)}$
    if the region given by the intersection of elements of S is empty **then**
      return Message indicating that the region is empty, and specify the current index ℓ where we reached an empty set
  return S

Algorithm 2 A tree traversal approach to finding all linear regions. Initial call is TRAVERSE(0,$\varnothing$).

```
results ← a global array, accessible within nested calls
function TRAVERSE(depth, v)
        ▷ v is some set of elements {v^(1), ..., v^(k)}. Initially, v is empty.
    if depth == L then
        msg ← FINDLINEARREGION(v)                    ▷ Call algorithm 1.
        if msg shows an empty region then
            return index of failure
        else
            Append msg containing linear region to global array of results
            return nothing
    else
        for each possible assignment of −1 and +1 to entries of v_depth
        do
            v′ ← current assignment of −1 and +1
            msg ← TRAVERSE(depth+1, v ∪ v′)          ▷ Recursive call.
            if msg contains index of a failure, and current depth > index
            of failure then
                return index of failure
```

Algorithm 3 Algorithm to make predictions and cache observed linear regions.

```
function PREDICT(x)
    Find linear the region v ∈ {−1, 1}^num_neurons in which x lies.
    ▷ This can be achieved by checking the inequalities that define each
    linear region.
    if v is not in the cache then
        Use recurrence relations defined in (14) to find the matrix F_A^(L)
        and vector f_b^(L) s.t. the neural network map is F_A^(L) x + f_b^(L) on
        the given linear region.
        Append the key-value pair (v : (F_A^(L), f_b^(L))) to the cache
    Use the cache to look up F_A^(L), f_b^(L) for the current linear region
    return F_A^(L) x + f_b^(L)
```

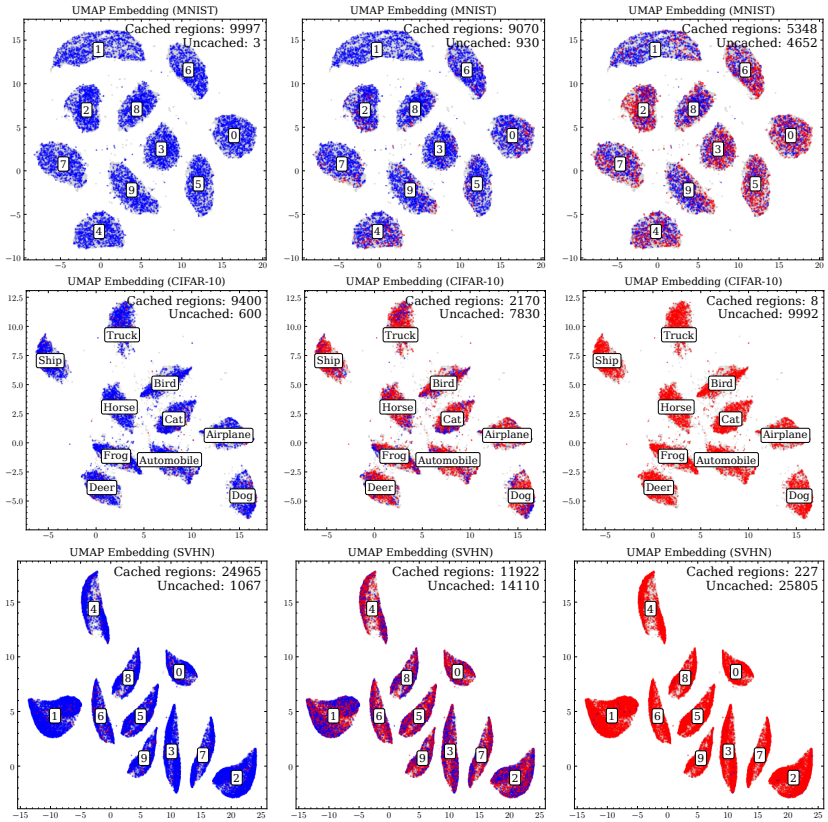

Figure 4: UMAP embeddings showing rates of caching in the test set for each of; MNIST *(top row)*, CIFAR-10 *(middle row)*, and SVHN *(bottom row)*. Each column represents a different model size; 4 layers and 8 neurons per layer *(left column)*, 3 layers and 16 neurons per layer *(middle column)*, and 3 layers and 32 neurons per layer *(right column)*. Blue points denote test data in the same linear region as at least 1 training set instance, while red points are *not* in the same region as any training instance. Gray points represent training data. Labels show the mean coordinates of each class.