A PROOF OF THEOREM 1

We describe the proof of Theorem 1 in two major steps as follows. We will repeat some of the steps in the sketch of the proof for completeness.

Step-1: By an " input configuration," we mean a certain subset of neurons that are active at a certain layer of the unconditional "base" network. We represent input configurations by binary vectors where "0" represents an inactive neuron, while "1" represents an active neuron. As an example, the input configuration $[1\,0\,1]^T$ refers to a scenario where only the first and the third neurons at the layer is active. We follow the convention that all components of the input vectors are "active" nodes.

Analogous to input configurations, "output configurations" are binary vectors that represent whether neurons provide zero or non-zero outputs. For example, an output configuration of $[1\,1\,0]^T$ means that only the first and the second neurons provide a non-zero output. At Layer 0, or the input layer, we follow the convention that the output configuration represents whether the components of the input are zero or non-zero.

For our purposes, the significance of input and output configurations is the following observation: Suppose that, at Layer ℓ of an unconditional network, a particular input y_0 provides the input and output configurations $\texttt{IC}_\ell$ and $\texttt{OC}_\ell$, respectively. In particular, we have $\texttt{OC}_0 = \mathbf{1}(y_0 \neq 0)$. According to the neural network input-output relationship in (1), and the definition of input and output configurations, we have the identity

$$\texttt{IC}_\ell = \mathbf{1}\left(\mathbf{1}(W_\ell \neq 0)[\texttt{OC}_{\ell-1}^T \cdots \texttt{OC}_0^T]^T \neq 0\right), \tag{6}$$

where $\mathbf{1}(\cdot)$ represents the indicator function, applied component-wise. In other words, knowing the output configurations of preceding Layers $0, \ldots, \ell-1$, we can uniquely determine the input configuration of Layer ℓ. To see why (6) holds, note that some Neuron j at Layer ℓ is active (and thus $\texttt{IC}_{\ell,j} = 1$) if it admits at least one non-zero input from any one of the previous layers, multiplied by a corresponding non-zero weight in W_ℓ. The indicator functions encode this criterion.

Exploiting the above observation that output configurations imply unique input configurations, the first step of the proof is thus to construct what we call a "configuration tree." The configuration tree is a directed rooted tree, where nodes at Depth ℓ correspond to all possible sequences of output configurations from neural network Layers $0, \ldots, \ell-1$, which imply a unique input configuration at Layer ℓ. We only need to consider the sequences induced by the training set. We represent vertices of the tree by the sequence of output configurations and the unique input configuration that is induced by them (The inclusion of the induced input configuration to the vertex information is thus superfluous and mainly to make the exposition clearer.). The only exception is the root of the tree, which is represented by an all-1 vector of dimension p. This corresponds to the unique input configuration of Layer 0 (all components of the input are considered "active."). Edges between vertices represent an output configuration that maps the sequence of output configurations to an input configuration.

We construct the configuration tree by showing the inputs to the neural network one by one. We begin with the first element x_1 of the dataset. The corresponding input configuration $\texttt{IC}_0 = [1 \cdots 1]^T$ at Layer 0 is already available on the tree as the root node. As the next step, we determine the output configuration $\texttt{OC}_0 = \mathbf{1}(x_1 \neq 0)$ at Layer 0. Knowing the output configuration $\texttt{OC}_0$, we can uniquely determine the input configuration $\texttt{IC}_1 = \mathbf{1}(\mathbf{1}(W_1 \neq 0)\texttt{OC}_0 \neq 0)$ at Layer 1. We add $(\texttt{OC}_0, \texttt{IC}_1)$ as a new vertex to Depth 1 of the tree. We create an edge with label $\texttt{OC}_0$ to connect the root node $\texttt{IC}_0$ to $(\texttt{OC}_0, \texttt{IC}_1)$. We continue by finding the output configuration at Layer 1. Knowing the output configurations $\texttt{OC}_1$ and $\texttt{OC}_0$, we can uniquely determine the input configuration $\texttt{IC}_2$ at Layer 2, which we add as the vertex $(\texttt{OC}_1, \texttt{OC}_0, \texttt{IC}_2)$ to Depth 2. We create an edge with label $\texttt{OC}_1$ to connect $(\texttt{OC}_0, \texttt{IC}_1)$ to $(\texttt{OC}_1, \texttt{OC}_0, \texttt{IC}_2)$. We continue in the same manner for all subsequent layers, and then for all samples in the dataset.

The formal construction of the configuration tree is provided in Algorithm 2. Edges are defined as triplets, where the first two components are the endpoints, and the last component is the edge label.

Algorithm 2 An algorithm to construct the configuration tree

$\mathcal{V} \leftarrow \{[1 \cdots 1]^T\}$. ▷ Initialize the tree with a root node but no edges.
$\mathcal{E} \leftarrow \emptyset$.
for $i = 1$ to n **do**
  $y_0 \leftarrow x_i$ ▷ A dataset input to the neural network.
  $\overline{\texttt{OC}}_{-1} = \emptyset$ ▷ A convention specific to this algorithm.
  $\texttt{IC}_0 \leftarrow [1 \cdots 1]^T$. ▷ All input components are active by convention.
  for $\ell = 1$ to L **do**
    $\overline{y}_{\ell-1} \leftarrow [y_{\ell-1}^T \cdots y_0^T]^T$. ▷ Outputs of all Layers $< \ell$ are inputs to Layer ℓ.
    $\overline{\texttt{OC}}_{\ell-1} \leftarrow \mathbf{1}(\overline{y}_{\ell-1} \neq 0)$. ▷ Output configurations of all inputs to Layer ℓ.
    $\texttt{IC}_\ell \leftarrow \mathbf{1}(\mathbf{1}(W_\ell \neq 0)\overline{\texttt{OC}}_{\ell-1} \neq 0)$. ▷ Same formula as (6).
    $\mathcal{V} \leftarrow \mathcal{V} \cup \{(\overline{\texttt{OC}}_{\ell-1}, \texttt{IC}_\ell)\}$.
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\overline{\texttt{OC}}_{\ell-2}, \texttt{IC}_{\ell-1}), (\overline{\texttt{OC}}_{\ell-1}, \texttt{IC}_\ell), \texttt{OC}_{\ell-1})\}$.
    $y_\ell \leftarrow \phi(W_\ell \overline{y})$.
    $\texttt{OC}_\ell \leftarrow \mathbf{1}(y_\ell \neq 0)$.
  end for
end for

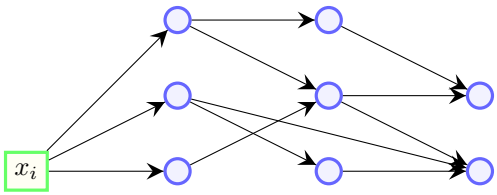


Figure 3: An example network for demonstrating the construction of the configuration tree.

Example 1. As an example of the construction of the configuration tree, consider the network in Fig. 3 with one dimensional inputs and two dimensional outputs. We consider a dataset with two samples $n = 2$. Suppose that the first input $x_1 \neq 0$ provides the input configurations [1 1 1], [1 1 0], [1 0], and output configurations [1 0 1], [1 0 0], [1 0], at Layers 1, 2, 3, respectively. Then, after processing x_1, the tree will consist of all blue/white nodes in Fig. 4 and the edges between them. Note that the last components of all vertices represent the input configurations, while edges represent the output configurations. Hence, the node 101, 1, 110 at Depth 2 of the tree represents the scenario where only the first and the second neurons in the second layer are active (have at least one non-zero input). Since there are 2 active neurons, there are 4 possible output configurations: 000, 010, 011, and 110. In particular, the output configuration 010 implies an input configuration of 11 at Depth 3 (Layer 3). This is because, Node 2 of Layer 2 is connected to both neurons at Layer 3, and a non-zero output of this neuron implies a non-zero input to Layer 3 neurons. Now, suppose the second input $x_2 \neq 0$ provides the input configurations [1 1 1], [0 1 0], [1 1], and output configurations [0 0 1], [0 1 0], [1 1], at Layers 1, 2, 3, respectively. Then, the construction of the tree is completed as shown in Fig. 4.□

Step-2: We now construct the conditional network itself out of the configuration tree. In fact, the conditional network will follow the same structure as the configuration tree, traversing from the root of the tree to one of the leaves as we calculate the output for a given input to the neural network. As in the case of the configuration tree, we calculate the layer outputs sequentially one at a time. First, we find the output of Layer 1. Given any input y_0 to the network, the input configuration $\texttt{IC}_0$ at Layer 0 is the all-one vector of dimension p. The conditional network first compares against all

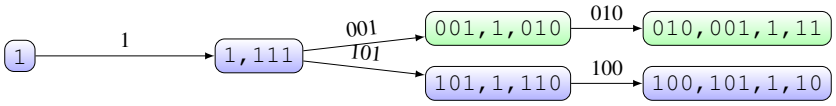


Figure 4: One configuration tree corresponding to the network in Fig. 3.

possible output configurations that are found at the configuration tree at Layer 0, which correspond to edges connected to the root node. Since the input dimension is p, this can be done through p nested binary conditions. Moreover, p operations are sufficient to reach the "leaves" of p nested conditions. An example of this step is provided in Algorithm 3 for a two-dimensional input $[u_1 u_2]^T$. In general, at each of the 2^p leaves of nested conditions, we uniquely know the output configuration $\texttt{OC}_0$ at Layer 0, as well as the input configuration $\texttt{IC}_1$ at Layer 1 via the configuration tree. At Layer 1, we only need to calculate the outputs of the neurons indicated by the input configuration $\texttt{IC}_1$, since other neurons are guaranteed to have all-zero inputs and thus provide zero output. This yields the output y_1 of the entire Layer 1. At this stage, we have obtained the input y_1 to the second layer (possibly in addition to y_0, the skip connections from inputs, that we already know), and the input configuration $\texttt{IC}_1$ at Layer 1. We proceed recursively in the same manner described above: That is to say, $|\texttt{IC}_1|$ nested binary conditions are utilized to determine the output configuration $\texttt{OC}_1$. Together with $\texttt{OC}_0$, this yields the input configuration $\texttt{IC}_2$ at Layer 2 as well as the Layer 2 outputs y_2. Outputs for all subsequent layers are determined in a similar fashion.

Algorithm 3 Two nested binary conditions

```
if u_1 = 0 then
    if u_2 = 0 then
        ...
    else
        ...
    end if
else
    if u_2 = 0 then
        ...
    else
        ...
    end if
end if
```

Algorithm 4 Construction of the conditional network out of the configuration tree. In the algorithm, x is the input to the neural network. Outputs are provided in the variable y_L.

1: Replace the root node with $y_0 \leftarrow x$.

2: Replace any edge of the form $((\overline{\texttt{OC}}_{\ell-2}, \texttt{IC}_{\ell-1}), (\overline{\texttt{OC}}_{\ell-1}, \texttt{IC}_\ell), \texttt{OC}_{\ell-1})$ with the conditioning statement

$$\text{If } \mathbf{1}(y_{\ell-1}[\![\texttt{IC}_{\ell-1}]\!] \neq 0) = \texttt{OC}_{\ell-1}[\![\texttt{IC}_{\ell-1}]\!], \text{ then} \tag{7}$$

3: Replace any vertex of the form $(\overline{\texttt{OC}}_{\ell-1}, \texttt{IC}_\ell)$ with

$$y_\ell[\![\texttt{IC}_\ell]\!] = \phi(W_\ell[\![\texttt{IC}_\ell, \overline{\texttt{OC}}_{\ell-1}]\!]\overline{y}_{\ell-1}[\![\overline{\texttt{OC}}_{\ell-1}]\!]) \,, \tag{8}$$

where $\overline{y}_{\ell-1} \triangleq [y_{\ell-1}^T \cdots y_0^T]^T$.

The construction of the conditional network is formally stated in Algorithm 4. In the description of the algorithm, for any vector $a = [a_1 \cdots a_n]$, and binary vector $b = [b_1 \cdots b_n] \in \{0,1\}^n$, we use the notation that $a[\![b]\!]$ to represent the $\|b\|_1$-dimensional vector consisting only of components a_i such that $b_i = 1$. Here, $\|\cdot\|_1$ is the 1-norm, counting the number of non-zero components for the case of a binary vector. Likewise, for a matrix W, the notation $W[\![b_1, b_2]\!]$ represents the $\|b_1\|_1 \times \|b_2\|_1$ matrix where we only consider those rows and columns of W as indicated by the non-zero components of b_1 and b_2, respectively. For example, $[6,9,7][\![1,0,1]\!] = [6,7]$, and $\left[\begin{smallmatrix} 3 & 6 & 8 \\ 1 & 4 & 7 \end{smallmatrix}\right][\![[0,1],[0,1,1]]\!] = [4,7]$. Note that the conditioning statements in Line 2 are actually implemented via nested binary conditions as described in Algorithm 3. We have presented the nested conditionings in their compact form shown in Line 2 for a clearer exposition.

Let us now show that the conditional network as constructed in Algorithm 4 satisfies the statement of the theorem. In particular, it provides the same output as the unconditional network over the dataset. This can be shown by induction. Suppose the network manages to correctly calculate all outputs and input/output configurations up to and including Layer $\ell - 1$ for a certain input with the exception of $\texttt{OC}_{\ell-1}$, which is yet to be determined. At the conditional network graph, we are then "at the end of" vertex $(\overline{\texttt{OC}}_{\ell-2}, \texttt{IC}_{\ell-1})$, having just executed (7) for $\ell \leftarrow \ell - 1$. A combination of (7) and (8) will now determine $\texttt{OC}_{\ell-1}$, $\texttt{IC}_\ell$, and y_ℓ. Suppose that the true values for these variables are $\mathbf{O}$, $\mathbf{I}$, and $\mathbf{y}$, respectively. We will thus show that $\texttt{OC}_{\ell-1} = \mathbf{O}$, $\texttt{IC}_\ell = \mathbf{I}$ and $y_\ell = \mathbf{y}$. We first note that in (7), we will have an edge with $\texttt{OC}_{\ell-1} = \mathbf{O}$. This follows as the conditional network is constructed out of the configuration tree, which itself is constructed from the dataset. Since $\mathbf{O}$ exists on the configuration tree, so it should on the conditional network tree. The conditional network then transitions to (8) with $\texttt{OC}_{\ell-1} = \mathbf{O}$ and $\texttt{IC}_\ell = \mathbf{I}$. Here, the input configuration is correctly calculated as it is unique given previous output configurations. It follows that (8) is calculated correctly so that $y_\ell = \mathbf{y}$.

We now evaluate the computational complexity. Note that, Line 2 of Algorithm 4 can be implemented via $|\texttt{IC}_{\ell-1}|_1$ nested binary conditions, requiring $|\texttt{IC}_{\ell-1}|_1$ operations. We now evaluate the cost of Line 3, where only the outputs of active nodes of Layer ℓ are calculated. Let a_ℓ denote the number of active nodes at Layer ℓ. Line 3 performs at most a_ℓ multiplications and a_ℓ additions to calculate the local fields of these active neurons. Calculating the active neuron outputs through ReLU activation functions require a further $|\texttt{IC}_\ell|_1$ comparisons or operations. Hence, Line 3 requires $2a_\ell + |\texttt{IC}_\ell|_1$ operations at most. Traversing from the root to a leaf of the computation graph, the total number of operations is at most $p + 2\sum_{\ell=1}^{L}(a_\ell + |\texttt{IC}_\ell|_1)$. Since, at each layer, there should be as many active neurons as there are active weights, we have $|\texttt{IC}_\ell|_1 \leq a_\ell$. Substituting this estimate to the previous bound, we obtain the same upper bound in the statement of the theorem. This concludes the proof of Theorem 1.

One aspect of the conditional network, as constructed in Algorithm 4 is that certain inputs that do not belong to the dataset may end up in conditions for which no output is defined. This occurs when one of the bins or vertices induced by the nested binary conditions in Line 2 (and as exemplified in Algorithm 3) remains empty. Since the main focus of this work is memorization of a dataset, we are not concerned with the network operation for such inputs. Nevertheless, the network operation can easily be generalized so that the output is well-defined for any input, e.g. simply by removing the binary condition with one or two empty bins, and reconnecting any children of the removed condition to the condition's parent.

B Proof of Theorem 2

We explicitly construct the neural network that achieves the performance as provided in the theorem statement. Our construction relies on several steps as described in the following:

Step-1: Let $u = [u_1\, u_2 \cdots u_q]^T$ be the input to the neural network, where $u_1 = 1$. Let $\bar{u} = [\bar{u}_1\, \bar{u}_2 \cdots \bar{u}_q]^T$ represent the output of the first layer. First, we translate the dataset vectors such that every component of the translated vectors is positive. We also provide a skip connection for the constant input 1 at the first component. For the former purpose, given $j \in \{1 \ldots, p\}$, let x_{ij} denote the jth component of dataset pattern x_i. We define the constant

$$M = 1 + \max_{i\in\{1,\ldots,n\}} \max_{j\in\{1,\ldots,p\}} |x_{ij}|. \tag{9}$$

The input-output relationships of the first layer is then expressed as

$$\bar{u}_1 = u_1 = 1. \tag{10}$$

$$\bar{u}_j = \phi\left(\begin{bmatrix} u_1 \\ u_j \end{bmatrix}^T \begin{bmatrix} M \\ 1 \end{bmatrix}\right),\ j = 2, \ldots, q. \tag{11}$$

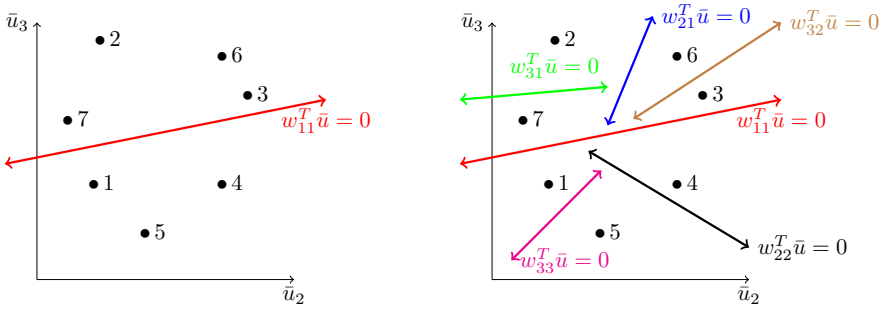
In particular, if $u = \bar{x}_i$ for some $i \in \{1, \ldots, n\}$, then

$$\bar{u}_j = \phi(u_j + M) = u_j + M,\ j = 2, \ldots, q. \tag{12}$$

The last equality follows as $u_j + M = x_{ij} + M$ is positive by the definition of M. Hence, for the dataset members, the output after the first layer of neurons is an additive translation by the vector $[0\, M \cdots M]^T$. There are $p = q - 1$ active neurons in the first layer.

Step-2: For a clearer exposition, we first describe the step through an example. We follow a divide and conquer strategy resembling a binary search, which is illustrated in Fig. 5. Suppose that the inputs are two-dimensional $p = 2$ and the outputs are scalars $q = 1$. In Fig. 5a, we show the second and the third components $[\bar{u}_2\, \bar{u}_3]^T$ of 7 dataset patterns after the first layer, together with the indices of the patterns. We also show a line $w_{11}^T \bar{u} = 0$ that separates the set of points to two subsets such that one side of the line contains $\lceil \frac{7}{2} \rceil = 4$ points, and the other side of the line contains the remaining 3 points. We will formally show later that, given m points, a separation where one side of the hyperplane contains exactly $\lceil \frac{m}{2} \rceil$ of the points is always possible in general.

Once an even division or separation is achieved, the next step is to design what we call a "switching network". Switching networks are small two-layer neural networks that can be parametrized by any number of points greater than 1. Let 0_q denote the q-dimensional all-zero column vector. Roughly speaking, for the scenario in Fig. 5a, the corresponding switching network maps the input $\bar{u} \in \mathbb{R}^3$ to



(a) Dividing a set of point to two equal subsets. (b) Continued divisions until reaching singletons.

Figure 5: The divide and conquer strategy. This figure is a duplicate of Fig. 1 in order to make the proof easier to follow.

$\left[\begin{smallmatrix}\bar{u}\\0_3\end{smallmatrix}\right] \in \mathbb{R}^6$ if $\bar{u}$ remains "above" the line $w_{11}^T\bar{u}=0$, and to $\left[\begin{smallmatrix}0_3\\\bar{u}\end{smallmatrix}\right]$ if $\bar{u}$ remains "below" the line.[2] We can now feed the first 3 components of the output of the switch to one subnetwork, and the last 3 components to another subnetwork. The two subnetworks are disconnected except that they share different components of the same output as inputs. The first subnetwork follows the same divide and conquer strategy with a switch, but only for the four points that remain over the line $w_{11}^T\bar{u}=0$. The second subnetwork similarly processes the three points that remain under the line $w_{11}^T\bar{u}=0$. The goal of the all-zero outputs is to deactivate one subnetwork when it is no longer relevant to process a given input. Subnetworks have "subsubnetworks" and so on until one arrives at a singleton dataset sample at each partition, as shown in Fig. 5b.

Before proceeding to the next step, we formalize the constructions in Step 2 via the following lemma.

Lemma 1. *For $m \geq 2$, let $a_1, \ldots, a_m \in \mathbb{R}^q$ be distinct input patters whose first components equal 1.*

[i] There exists $w \in \mathbb{R}^q$, such that

$$|\{i : w^T a_i < 0\}| = \lceil \tfrac{m}{2} \rceil, \tag{13}$$

$$|\{i : w^T a_i > 0\}| = m - \lceil \tfrac{m}{2} \rceil. \tag{14}$$

[ii] Suppose further that the components of a_is are all non-negative. Let $w \in \mathbb{R}^q$ satisfy (13) and (14). Let 0_q represent the q-dimensional all-zero vector. There is a two-layer network $\mathbf{S} : \mathbb{R}^q \to \mathbb{R}^{2q}$ that satisfies the input-output relationships

$$\mathbf{S}(a_i) = \begin{cases} \begin{bmatrix} a_i \\ 0_q \end{bmatrix}, & w^T a_i < 0, \\ \begin{bmatrix} 0_q \\ a_i \end{bmatrix}, & w^T a_i > 0. \end{cases}, \; i = 1, \ldots, m. \tag{15}$$

The network has $2q+2$ neurons. Two of the $2q+2$ neurons have q weights, and the remaining $2q$ neurons have 2 weights.

Proof. Let us first prove [i]. Let $\bar{a}_i = [a_{i,2}, \ldots, a_{i,q}]^T$ denote the $(q-1)$-dimensional vector consisting of all but the first component of a_i (which equals 1). First, we show the existence of $\bar{w} \in \mathbb{R}^{q-1}$ such that $\bar{w}^T\bar{a}_i \neq \bar{w}^T\bar{a}_j$, $\forall i \neq j$, or equivalently

$$\bar{w}^T(\bar{a}_i - \bar{a}_j) \neq 0, \; \forall i \neq j. \tag{16}$$

Since the input patterns a_is are distinct, so are $\bar{a}_i$s, and thus $\bar{a}_i - \bar{a}_j$ are non-zero vectors. It follows that any unit norm $\bar{w}$ sampled uniformly at random on $\{x \in \mathbb{R}^q : \|x\| = 1\}$ satisfies (16) with probability 1. Let us now order the resulting $(\bar{w}^T\bar{a}_i)$s in ascending order as

$$\bar{w}^T\bar{a}_{i_1} < \cdots < \bar{w}^T\bar{a}_{i_{\lceil \frac{m}{2} \rceil}} < \bar{w}^T\bar{a}_{i_{\lceil \frac{m}{2} \rceil + 1}} < \cdots < \bar{w}^T\bar{a}_{i_m}, \tag{17}$$

[2] More precisely, there will be a small margin around the separating hyperplane where the aforementioned input-output relationships may fail. It turns out that this technical complication poses no issues for the patterns that we wish to memorize, as the margin can be made arbitrarily small.

for some permutation $i_1, \ldots, i_m$ of $1, \ldots, m$. We can now tune the bias as

$$\bar{\bar{w}} \triangleq -\frac{1}{2}\left(\bar{w}^T \bar{a}_{i_{\lceil \frac{m}{2} \rceil}} + \bar{w}^T \bar{a}_{i_{\lceil \frac{m}{2} \rceil + 1}}\right). \tag{18}$$

The effect of the bias is the ordering (note the zero in the middle)

$$\bar{\bar{w}} + \bar{w}^T \bar{a}_{i_1} < \cdots < \bar{\bar{w}} + \bar{w}^T \bar{a}_{i_{\lceil \frac{m}{2} \rceil}} < 0 < \bar{\bar{w}} + \bar{w}^T \bar{a}_{i_{\lceil \frac{m}{2} \rceil + 1}} < \cdots < \bar{\bar{w}} + \bar{w}^T \bar{a}_{i_m}. \tag{19}$$

Therefore, the choice $w = \left[\begin{smallmatrix} \bar{\bar{w}} \\ \bar{w} \end{smallmatrix}\right]$ satisfies conditions (13) and (14). This concludes the proof of [i].

We now prove [ii]. Let

$$C_1 \triangleq 1 + \max_{i \in \{1,\ldots,m\}} \max_{j \in \{1,\ldots,q\}} |a_{ij}|, \tag{20}$$

$$C_2 \triangleq \min_{i \in \{1,\ldots,m\}} |w^T a_i|. \tag{21}$$

Let $v = [v_1 \cdots v_q]^T \in \mathbb{R}^q$ represent an input to the neural network $\mathbf{S}$ that we shall construct. Also, let $v^+ = [v_1^+ \cdots v_q^+]^T$ and $v^- = [v_1^- \cdots v_q^-]^T \in \mathbb{R}^q$ denote the first and the last q components of the $2q$-dimensional output of $\mathbf{S}$. We thus have $\mathbf{S}(v) = \left[\begin{smallmatrix} v^+ \\ v^- \end{smallmatrix}\right]$, and set

$$v_j^+ = \phi\left(\begin{bmatrix} v_j \\ y^+ \end{bmatrix}^T \begin{bmatrix} 1 \\ -\frac{C_1}{C_2} \end{bmatrix}\right),\ v_j^- = \phi\left(\begin{bmatrix} v_j \\ y^- \end{bmatrix}^T \begin{bmatrix} 1 \\ -\frac{C_1}{C_2} \end{bmatrix}\right),\ j \in \{1, \ldots, q\}, \tag{22}$$

where $y^+ = \phi(w^T v)$ and $y^- = \phi(-w^T v)$. Let us now verify (15). Consider some index i with $w^T a_i < 0$, and suppose $v = a_i$. For any $j \in \{1, \ldots, q\}$, we have

$$v_j^+ = \phi(v_j - \tfrac{C_1}{C_2} y^+) = \phi(v_j) = \phi(a_{ij}) = a_{ij}. \tag{23}$$

The last equality holds as the components of a_i are assumed to be all non-negative. Also,

$$v_j^- = \phi(v_j - \tfrac{C_1}{C_2} y^-) = \phi(a_{ij} - \tfrac{C_1}{C_2} |w^T a_i|) \leq \phi(a_{ij} - C_1) = 0. \tag{24}$$

Inequality follows as $\frac{|w^T a_i|}{C_2} \geq 1$ and ϕ is monotonic. Since $v_j^- \geq 0$ obviously holds, we have $v_j^- = 0$. This proves the case $w^T a_i < 0$ in (15). The remaining case $w^T a_i > 0$ can be verified in a similar manner. This concludes the proof of [ii], and thus that of Lemma 1 as well. □

Step-3: We can now proceed to describe the full network architecture, as shown in Fig. 6 for the example in Step 2. The first layer of the network is an additive translation by the vector $[0\, M \cdots M]^T$, and is explained in Step 1 above. We use the notation $\mathbf{T}$ to denote the translation, which is followed by a sequence of switches as described in Step 2. In particular, $\mathbf{S}_{ij}$ provides the outputs of $\bar{u}$ and 0_3 to its top and bottom branches, respectively, if its input $\bar{u}$ remains above the line defined by w_{ij} in Fig. 5b. By construction, the neurons on a given path of switches is activated for a unique dataset member. For example, the path $\mathbf{S}_{11}, \mathbf{S}_{21}, \mathbf{S}_{32}$ is activated only for the dataset member x_6. The path of switches that correspond to some x_i is followed by a ReLU neuron whose weights satisfy the property that $\phi(\gamma_i^T \bar{\bar{x}}_i) = d_i$, where $\bar{\bar{x}}_i$ is the output of the first layer when the input to the network is $\bar{x}_i$. Since the first component of $\bar{\bar{x}}_i$ equals 1 for any i, one can simply set the first component of γ_i to be equal to d_i, and the rest of the component of γ_i to be equal to zero. The final layers of the network simply adds all the outputs from the γ_i-neurons.

In the figure, we also show the induced signals on the branches when the input is $u = \bar{x}_6$ as an example. Note that only the block $\mathbf{T}$, switches $\mathbf{S}_{11}, \mathbf{S}_{21}, \mathbf{S}_{32}$, the neuron with weight γ_6^T, and the a subset of the summation neurons remains active. Most of the neurons of the network are deactivated through zero signals. We also note that the desired output signal d_6 is obtained.

The construction generalizes to an arbitrary dataset of cardinality n in the same manner. The only difference is that, in order to support r-dimensional desired outputs as stated in the theorem, we need to use r ReLU units in place of each γ_i to reproduce the r components of the desired output, as opposed to a single ReLU unit in the example above. Also, the final summation unit will consist of r sub-summation units operating on individual components. Formally, the first layer for the general case is the translation $\mathbf{T}$ as before. Next, $\lceil \log_2 n \rceil$ layers of switches arranged on a binary tree structure

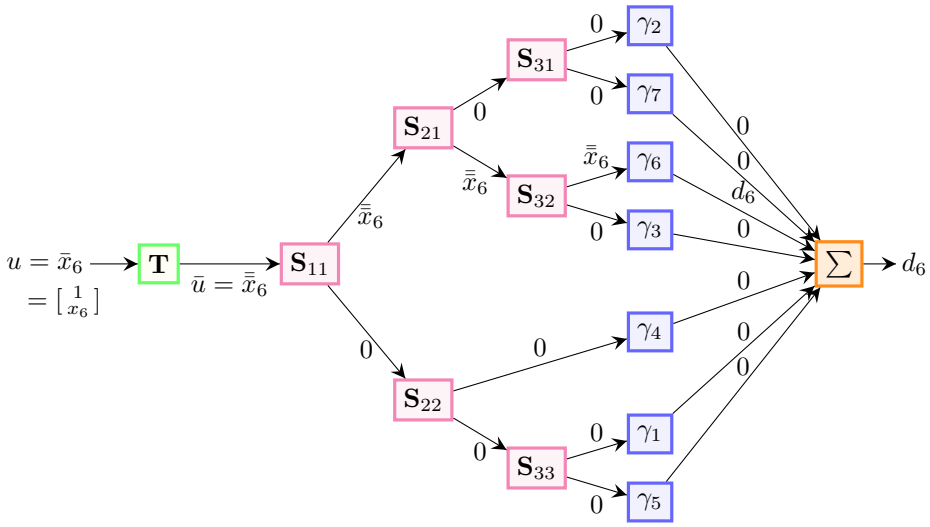


Figure 6: An example network architecture for the achievability result. The block $\mathbf{T}$ represents the transformation in Step 1. Blocks $\mathbf{S}_{ij}$ are the routing switches. Blocks γ_i represent ReLU neurons with weights γ_i, and the $\sum$ block represents a ReLU neuron with all-one weights. This figure is a duplicate of Fig. 2 in order to make the proof easier to follow.

act on the translated inputs, forming n leaves as the output, where each leaf has the same dimension as the input. Let $h_1, \ldots, h_n$ denote the feature vectors produced at the leaf nodes after the switches. By Lemma 1, and a rearrangement of indices, we can guarantee that for every i, if the network input is $\bar{x}_i$, then $h_i = \bar{\bar{x}}_i$ and $h_j = 0$, $\forall j \neq i$. Here, $\bar{\bar{x}}_i = \mathbf{T}([1\ x_i^T]^T)$, as defined in Fig. 6. Each leaf is then followed by a single-layer network that can map $\bar{\bar{x}}_i$ to its corresponding desired output vector d_i. Specifically, there exists U_i such that $d_i = \phi(U_i \bar{\bar{x}}_i)$. Indeed, since $\sum_k u_{i,j,k} \bar{\bar{x}}_{i,k} = d_{i,j}$ has to be satisfied, one can pick $u_{i,j,k} = 0$, $k \neq 1$ and $u_{i,j,1} = d_{i,j} / \bar{\bar{x}}_{i,1} = d_{i,j}$, where the last equality holds as $\bar{\bar{x}}_{i,1} = 1$ by construction. The overall network output is the accumulation of the outputs of the U_i-neurons, and is given by $f(\bar{x}_i) = \phi(\sum_{i=1}^n \phi(U_i h_i))$. Let us show that all input-output relationships are satisfied so that the claim $f(\bar{x}_i) = d_i$, $\forall i$ in the statement of the theorem holds. If the input is $\bar{x}_i$, we have $h_i = \bar{\bar{x}}_i$ and $h_j = 0$, $j \neq i$, by construction. As a result, $f(\bar{x}_i) = \phi(\phi(U_i \bar{\bar{x}}_i)) = \phi(d_i) = d_i$, as desired. The last equality holds as the components of d_i are assumed non-zero in the statement of the theorem (A ReLU network cannot provide a negative output.).

Let us now calculate the number of active neurons and weights when the input belongs to any member of the dataset. The block $\mathbf{T}$ always remains active and consists of $q - 1$ neurons with weight 2. There are at most $\lceil \log_2 n \rceil$ active switches per input. Each switch contains $2q + 2$ neurons with $6q$ weights total. There is one active block γ_i consisting of r neurons with q weights each. Finally, the sum unit has r active weights. Hence, an upper bound on the total number of active neurons are given by

$$q - 1 + (2q + 2)\lceil \log_2 n \rceil + r + 1 = 2(q + 1)\lceil \log_2 n \rceil + q + r \tag{25}$$
$$\in O(r + q \log n), \tag{26}$$

and an upper bound on the number of active weights can be calculated to be

$$2(q - 1) + 6q(2q + 2)\lceil \log_2 n \rceil + rq + r = 12q(q + 1)\lceil \log_2 n \rceil + (r + 2)q + r - 2 \tag{27}$$
$$\in O(rq + q^2 \log n). \tag{28}$$

This concludes the proof of the theorem.