On the Feasibility of Single-Pass Full-Capacity Learning in Linear Threshold Neurons with Binary Input Vectors

Ruipeng Liu [*1] **Borui He** [*1] **Naveed Tahir** [*1] **Garrett Ethan Katz** [1]

Abstract

Known learning rules tend to fall near one of two extremes: single-pass associative learning with low complexity and capacity, and multi-pass iterative learning with high complexity and capacity. In this work we investigate the mathematical feasibility of learning rules that are both single-pass and achieve the theoretical upper bound on capacity. We consider a fairly broad family of learning rules we call "span rules," which include known rules such as Hebbian learning, perceptron learning, and backpropagation as special cases. To our knowledge, previous work has not determined whether single-pass, full-capacity span rules exist, even in the most fundamental case of a linear threshold neuron with binary input vectors, which is the focus of this study. We derive a necessary condition for the existence of such learning rules, which takes the form of a linear program, and show that the linear program is infeasible. This establishes an impossibility result that span rules can not be both single-pass and full-capacity.

1. Introduction

Classical associative learning rules for linear threshold neurons, such as linear associative networks (Anderson, 1972; Kohonen, 1972) and Hopfield networks (Hopfield, 1982), can store training data with very low computational complexity. They require only one pass over the training examples, and each weight update calculates the new weights as a simple linear combination of the old weights and the current example. Their main disadvantage is their low storage capacity: The number of examples they can effectively store is no greater than the input dimension.

[*]Equal contribution [1]Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, New York, USA. Correspondence to: Garrett Ethan Katz <gkatz01@syr.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

On the other hand, iterative methods such as perceptron learning (Rosenblatt, 1958) and support vector machines (Boser et al., 1992) have the capacity to fit any linearly-separable data (Elizondo, 2006). However, they may require multiple passes over the data and have higher computational complexity. More recent approaches to associative memory (Krotov & Hopfield, 2016; Demircigil et al., 2017) and single-pass learning (Liu & Xu, 2016) fall between these two extremes, trading some amount of computational efficiency for higher capacity, or vice versa.

This paper investigates the feasibility of single-pass learning rules with the low computational complexity of classical associative methods, but the full capacity of iterative methods. As a starting point, we focus on the simplest case of a single linear threshold neuron with binary input vectors. To our knowledge, even for this case, previous work has not yet determined whether such learning rules can exist. Specifically, we consider learning rules satisfying two constraints: **(i)** The new weights must lie in the span of the old weights and the current example (which holds for many known learning rules), and **(ii)** the new weights must immediately fit the current example without changing the neuron's output on previous examples, as long as the examples remain linearly separable. Constraint **(i)** defines the family of learning rules we consider, and **(ii)** is the single-pass, full-capacity requirement. We investigate the mathematical feasibility of rules satisfying **(i)** and **(ii)**, deriving a necessary condition for their existence which takes the form of a linear program. It turns out that this linear program is feasible for input dimension $N \leq 7$, but infeasible for $N \geq 8$. Therefore, it is mathematically impossible for learning rules in the family we consider to be both single-pass and full-capacity.

2. Learning Model

We consider linear threshold functions over the vertices of the N-dimensional hypercube defined by

$$\phi(w, x) = \text{sign}(w^\top x), \tag{1}$$

where ϕ models a neuron with weight vector $w \in \mathbb{R}^N$ receiving binary input vector $x \in \{-1, +1\}^N$. Since $\phi(w, -x) = -\phi(w, x)$, the neuron's output on one half of the hypercube fully determines its output on the other

half. Therefore we limit our attention to the half cube

$$H^N = \{x \in \{-1,+1\}^N \quad : \quad x[1] = -1\}, \qquad (2)$$

where $v[i]$ denotes the i^{th} component of a vector v using 1-based indexing. This avoids redundancy and also encapsulates non-zero firing thresholds, since $x[1]$ is constant and $w[1] \cdot x[1] = -w[1]$ therefore acts as a bias.

A learning process for ϕ involves an initial weight vector $w^{(0)}$ and a stream of training examples over time. At time-step $t \geq 1$, an input $x^{(t)} \in H^N$ is presented with a target output or "label" $y^{(t)} \in \{-1,+1\}$. In response, the weights are updated from $w^{(t-1)}$ to $w^{(t)}$. We focus on learning rules where the new weights lie in the span of the old weights and the current input, which we call "span rules," i.e.:

Definition 2.1. A learning rule for ϕ is a ***span rule*** if it can be written in the form

$$w^{(t)} = \alpha^{(t)} w^{(t-1)} + \beta^{(t)} x^{(t)}, \qquad (3)$$

where $\alpha^{(t)} > 0$ and $\beta^{(t)} \in \mathbb{R}$ are scalars that may depend on $w^{(t-1)}$, $x^{(t)}$, and $y^{(t)}$.

Many known learning methods are span rules, including perceptron learning, Widrow-Hoff learning (Widrow & Hoff, 1960), and backpropagation (Linnainmaa, 1970; Rumelhart et al., 1985), where $\alpha \in (0,1)$ is used to model weight decay. Span rules also relate to representer theorems, which in their simplest form state that an optimal solution to a learning problem is a linear combination of inner products with the training examples (Schölkopf et al., 2001). However, these theorems do not address whether the linear combination coefficients can be computed online in a single pass.

We say a rule is single-pass for a given training set if, after one presentation of each example, the resulting weights fit all of the examples. We say that the single-pass rule is also full-capacity if it is single-pass for every linearly separable training set. Formally:

Definition 2.2. A learning rule for ϕ is ***single-pass, full-capacity*** provided that, for any linearly separable training stream $\langle (x^{(1)}, y^{(1)}), ..., (x^{(t)}, y^{(t)}) \rangle$, the resulting weights $w^{(t)}$ at time-step t satisfy

$$\forall\, 1 \leq s \leq t \quad \phi(w^{(t)}, x^{(s)}) = y^{(s)}. \qquad (4)$$

To simplify notation and derivations in this paper, we restrict our attention to finite training streams with no repeated examples, i.e., $t \leq |H^N|$ and $x^{(s)} \neq x^{(t)}$ for $s \neq t$. A more general formulation could allow an infinite data stream in which the same input vector x can appear more than once, potentially even with different labels at different times: For example, if the task changes or some associations should eventually be "overwritten" with new information. In this formulation, if we still require the neuron to immediately fit each new example without changing its response on any other inputs, then it is still single-pass and a harder learning problem than standard multi-pass learning. The finite stream assumption does not weaken our results, because if the finite stream variant is infeasible (as we will show), then certainly the more general variant is infeasible also.

3. Related Work

3.1. Learning Model Capacity

Existing literature has defined capacity in multiple ways. In the context of associative memory, "***storage capacity***" is the maximum number of training examples that a model can reliably store and recall. For N-dimensional input vectors, storage capacity of the Hopfield model is approximately $0.14N$ (Amit et al., 1985), although higher capacity is possible with other learning procedures and training distributions. Storage capacity is exactly N when the input vectors are orthogonal and stored with linear associative learning, or linearly independent and stored using the pseudo-inverse (Kohonen, 2012). It can be much larger when the input vectors are correlated (Gardner, 1988). For input vectors in general position, the average number of examples that can be linearly separated is $2N$, irrespective of the learning rule used to find the decision surface (Cover, 1965). Modern Hopfield networks have very high storage capacity, but equally high computational complexity, since all training examples must be stored explicitly and accessed by the recall process (Krotov & Hopfield, 2016; Demircigil et al., 2017).

"***Cardinal capacity***" measures the number of distinct functions a model architecture can represent, irrespective of learning rule (Baldi & Vershynin, 2019). For the linear threshold model ϕ, this is precisely the number of linearly separable dichotomies of H^N, where a "***dichotomy***" is a partition of H^N into two sets: one containing examples assigned $y = +1$, and the other containing examples assigned $y = -1$. This number of linearly separable dichotomies is known to be on the order of $2^{N^2(1+o(1))}$ (Zuev, 1989). We focus on full cardinal capacity in this paper, as it constitutes the theoretical limit on ϕ's data-fitting capability, irrespective of the learning rule used to fit the data.

3.2. Enumerating Linearly Separable Dichotomies

Enumerating the number of distinct linearly separable dichotomies of H^N is a well-studied problem in linear threshold logic (Chow, 1961; Ojha, 2000; Picton, 2016; Rao & Zhang, 2018). Winder (1966) gave a combinatoric procedure that determines the precise number for any N. This body of work characterizes the number of such dichotomies, but not how they relate to any particular learning rule.

Any dichotomy may be checked for linear separability by

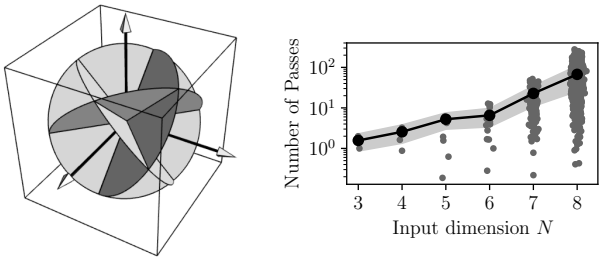

Figure 1. **Left:** Regions in weight space for each linearly separable dichotomy of H^3. Arrows are coordinate axes and shaded disks are nullplanes of each vertex x of the cube (black wireframe). **Right:** Passes to convergence (log scale) vs. input dimension N for the perceptron, one dot per canonical region (small noise added for legibility). Black line and gray envelope show mean and standard deviation, weighted by region equivalence class sizes.

solving a linear program for w. Specifically, the constraint $\phi(w, x) = y$ is equivalent to the linear inequality $w^\top xy > 0$, since $w^\top x$ has the same sign as y when their product is positive. Geometrically, the nullplane of each x separates $\mathbb{R}^N$ into two half-spaces, and a constraint $w^\top xy > 0$ confines w to one of those two half-spaces. When constraints for every (x, y) in a dichotomy are combined, w is confined to a polyhedral cone region. Therefore, when taken together, the nullplanes of the x's in H^N partition $\mathbb{R}^N$ into a set of regions, one per linearly separable dichotomy, as visualized in Figure 1 (left) for $N = 3$.

One can enumerate all the linearly separable dichotomies by checking feasibility of a series of such linear programs, which has been done for $N \leq 9$ (Winder, 1965; Muroga et al., 1970). A key ingredient in scaling to $N = 9$ was to leverage the symmetries of the hypercube, which correspond to the hyperoctahedral symmetry group (Todd, 1931). In particular, given a weight vector w in one region, w's in many other regions can be identified by permuting the entries of w and/or changing their signs (Goto & Takahasi, 1962). Regions related in this way can be grouped into equivalence classes (Slepian, 1953). For example, in Figure 1 (left), there are two equivalence classes: one for the three-sided regions, and one for the four-sided regions. Prior work only explicitly enumerates one representative region of each equivalence class, by imposing an additional (linear) constraint that the weight vectors contain non-negative entries in sorted order. These representative regions and their dichotomies are called "***canonical***."

3.3. Single- and Multi-Pass Learning

Single-pass learning is one instance of streaming algorithms, in which a data stream must be summarized efficiently without storing many datapoints in memory. Common applications of streaming algorithms include moment estimation (Alon et al., 1996), clustering (Charikar et al., 2003), and graph characterization (McGregor, 2014).

Linear associative networks (Anderson, 1972; Kohonen, 1972) and Hopfield networks (Hopfield, 1982) are classical single-pass learning models. They have also been called "one-shot," but in a different sense than recent "few-shot" methods, which focus on one or few examples per class rather than one presentation of each example (Vinyals et al., 2016; Brown et al., 2020; Lake et al., 2015; Fei-Fei et al., 2006). Classical single-pass models are quite efficient: Their weight updates tend to be span rules, in which the calculation of α and β is dominated by a single dot product. Hence their complexity tends to be $\mathcal{O}(N)$ per weight update. However, their main drawback is their low capacity.

Support vector machines (Boser et al., 1992) and perceptron learning (Rosenblatt, 1958; Murphy et al., 2017) are classical multi-pass learning methods. They achieve higher capacity by saving and accessing examples multiple times during training, which incurs space complexity linear in the size of the training data, and can also be very time-consuming. For example, perceptron convergence time is inversely related to a quantity called the "margin" of the training data (Block, 1962), which is exponentially small for some dichotomies of H^N in the worst and even average case. This is apparent in Figure 1 (right), where a separate perceptron was trained on every canonical dichotomy up to $N = 8$. Training generally required multiple passes over the "training examples," i.e., the vertices of H^N and their assigned labels. Before every pass of every training run, the examples were shuffled independently and identically at random. Training was halted as soon as every example was fit correctly (potentially midway through a pass).

Subsequent research has developed single-pass versions of support vector machines (Rai et al., 2009; Liu & Xu, 2016; Li & Long, 1999) and other learning models such as decision trees (Domingos & Hulten, 2000), the Winnow algorithm (Carvalho & Cohen, 2006), and orthogonal gradient descent (Min et al., 2022). Compared to iterative learning, these methods have lower running time, competitive empirical performance, and formal approximation error bounds. However, their learning rules tend to involve non-trivial algorithms that are substantially more complex than classical span rules. Hence, while existing methods strike a useful middle ground, they may not have achieved the theoretically optimal trade-off between capacity and complexity. The impossibility result contributed by this paper helps to characterize that theoretically optimal trade-off.

4. Theoretical Results

We conjectured that single-pass, full-capacity learning might be possible with a span rule, which turns out to be false. First we will formalize this conjecture, then we will derive a nec-

essary condition entailed by this conjecture. The necessary condition takes the form of a linear program, which we show to be infeasible. By contrapositive, this shows the conjecture to be false, which means that single-pass full-capacity learning is mathematically impossible using span rules.

To formalize the conjecture, we must consider all possible training streams. We can treat each possible stream as a different path through a large tree, where each edge corresponds to the presentation of a new example. The new example, and the new weights after the example is processed, are associated with the edge's destination node. For example, a small sub-graph of this tree is shown in Figure 2. Note that different nodes in the tree could contain the same training example, and that the sharing of nodes across paths precludes the weight updates from depending on downstream examples that have not been presented yet.

Formally, the set of possible length-T streams is $(H^N \times \{-1,+1\})^T$, since we choose one $x \in H^N$ and its label $y \in \{-1,+1\}$ at each of T time-steps. We arrange these streams in a tree $\mathcal{T}$, where the nodes at depth T are in one-to-one correspondence with the streams in $(H^N \times \{-1,+1\})^T$. For any node n with parent $p(n)$, the edge between them corresponds to presentation of one new training example, which we denote (x_n, y_n). We also let $\mathcal{D}_n$ denote the stream of all samples "seen so far" at node n, i.e.,

$$\mathcal{D}_n = \langle (x_{n_1}, y_{n_1}), ..., (x_{n_T}, y_{n_T}) \rangle, \tag{5}$$

where T is the depth of node n, and nodes $n_0, n_1, ..., n_T$ are the nodes along the path from the root to n, with n_0 being the root, $n_{T-1} = p(n)$, and $n_T = n$. We restrict our attention to the sub-graph $\overline{\mathcal{T}}$ of $\mathcal{T}$ in which each $\mathcal{D}_n$ is linearly separable, has length at most $|H^N|$, and contains no $x \in H^N$ more than once. Lastly, we let w_n denote the weight vector produced by a learning process after presenting the stream $\mathcal{D}_n$. At the root, w_0 represents the initial weights before learning begins.

Given this notation, the existence of a full-capacity, single-pass span rule can be formalize as:

Supposition 4.1. *There exist a set of vectors w_n and scalars $\alpha_n > 0$, $\beta_n \in \mathbb{R}$, such that for every non-root node n in $\overline{\mathcal{T}}$,*

$$\forall (x,y) \in \mathcal{D}_n \quad w_n^\top xy > 0, \tag{6}$$

$$w_n = \alpha_n w_{p(n)} + \beta_n x_n. \tag{7}$$

As mentioned earlier, condition (6) is equivalent to $\phi(w_n, x) = y$, but expressed as a linear inequality in w_n. Condition (7) is the span rule constraint.

We will show theoretically that Supposition 4.1 entails feasibility of a certain linear program, and then show that the linear program is infeasible. We find the infeasible linear program numerically, but also certify its infeasibility through a manual analysis. Consequently, Supposition 4.1 must be false by contrapositive, and we obtain the result that full-capacity, single-pass learning is not possible with a span rule.

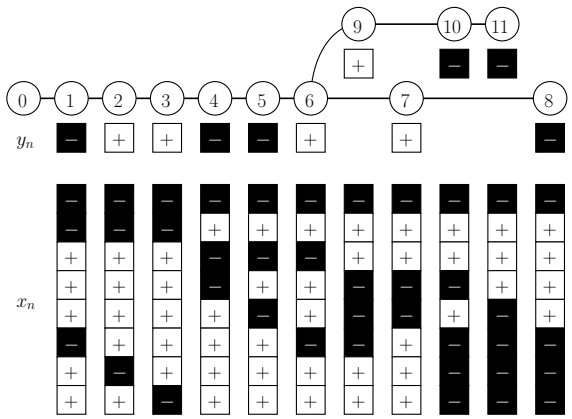

Figure 2. A small sub-graph of $\overline{\mathcal{T}}$ for $N = 8$. Circles are nodes with their index n written inside, and arcs are edges. The square immediately below each node n indicates the label y_n, and the column vectors further below indicate the input x_n. To reduce clutter we omit the "1" in each ± 1, only showing the sign.

First we remove the non-linear terms $\alpha_n w_{p(n)}$ with:

Proposition 4.1. *If Supposition 4.1 holds, then there exist a set of vectors u_n and scalars γ_n, such that for every non-root node n in $\overline{\mathcal{T}}$,*

$$\forall (x,y) \in \mathcal{D}_n \quad u_n^\top xy > 0, \tag{8}$$

$$u_n = u_{p(n)} + \gamma_n x_n. \tag{9}$$

Proof. Start by fixing the set of w_n, $\alpha_n > 0$, and β_n that exist under Supposition 4.1. Now for each node n in $\overline{\mathcal{T}}$ with path $n_0, ..., n_t$ from the root, where $n_t = n$, take

$$\gamma_n = \frac{\beta_n}{\prod_{s=1}^{t} \alpha_{n_s}}, \tag{10}$$

which is well-defined since all α's are non-zero. Next, construct u_n recursively according to (9), starting with $u_0 = w_0$. By constructing u_n this way, (9) is automatically satisfied, but we need to show that (8) is also satisfied. To do so, we will prove by induction that w_n is a positive scalar multiple of u_n for every n, so that our supposition $w_n^\top xy > 0$ in (6) entails our goal $u_n^\top xy > 0$ in (8). The base case holds at the root since we have assigned $u_0 = w_0$ in our construction of the u_n. For the inductive case, we will show that

$$w_n = \left(\prod_{s=1}^{t} \alpha_{n_s} \right) u_n, \tag{11}$$

for every non-root node n, which is a positive scalar multiple of u_n since all α's are positive. (11) holds because

$$w_{n_t} = \alpha_{n_t} w_{n_{t-1}} + \beta_{n_t} x_{n_t} \tag{12}$$

$$= \alpha_{n_t} \left(\prod_{s=1}^{t-1} \alpha_{n_s} \right) u_{n_{t-1}} + \beta_{n_t} x_{n_t} \tag{13}$$

$$= \left(\prod_{s=1}^{t} \alpha_{n_s} \right) \left(u_{n_{t-1}} + \gamma_{n_t} x_{n_t} \right) \tag{14}$$

$$= \left(\prod_{s=1}^{t} \alpha_{n_s} \right) u_{n_t}, \tag{15}$$

where (13) follows by the inductive hypothesis, (14) follows by factoring and the choice of γ_n in (10), and (15) follows by the construction of u_n according to (9). □

Next we note that if conditions (8) and (9) are true for all nodes in $\overline{\mathcal{T}}$, they are also true for any finite sub-graph $\hat{\mathcal{T}}$ of $\overline{\mathcal{T}}$. Given an arbitrary such $\hat{\mathcal{T}}$, let $\mathbb{P}(\epsilon)$ denote the linear program with parameter ϵ and variables u_n and γ_n given by

$$\min \sum_n \sum_{(x,y)\in\mathcal{D}_n} u_n^\top xy - \epsilon \quad \text{s.t.} \tag{16}$$

$$\forall n \, \forall (x,y) \in \mathcal{D}_n \quad u_n^\top xy \geq \epsilon \tag{17}$$

$$\forall n \neq 0 \quad u_n = u_{p(n)} + \gamma_n x_n. \tag{18}$$

The objective (16) minimizes the total slack in constraint (17) to ensure that the problem is bounded. $n \neq 0$ indicates non-root node. If there exist u_n and γ_n that satisfy (8) and (9), then they are also feasible solutions to $\mathbb{P}(\epsilon)$ when

$$\epsilon = \min_n \min_{(x,y)\in\mathcal{D}_n} u_n^\top xy. \tag{19}$$

Furthermore, dividing through by ϵ in (16-18) shows that if u_n and γ_n are feasible solutions to $\mathbb{P}(\epsilon)$, then u_n/ϵ and γ_n/ϵ are feasible solutions to $\mathbb{P}(1)$. Therefore, feasibility of $\mathbb{P}(1)$ for all $\hat{\mathcal{T}}$ is a necessary condition for Supposition 4.1 to hold. By contrapositive, if $\mathbb{P}(1)$ is infeasible for some $\hat{\mathcal{T}}$, then Supposition 4.1 must be false. The following sections confirm existence of one such $\hat{\mathcal{T}}$ in dimension $N = 8$.

Finally, we show that if constraints (8-9) are infeasible in a given dimension N, then they are also infeasible in dimension $N + 1$. Consequently, using $N = 8$ as a base case, infeasibility for all $N \geq 8$ follows by induction. In order to formally state this result, let $\overline{\mathcal{T}}_N$ denote the tree of all linearly separable streams in dimension N. We say that $\overline{\mathcal{T}}_N$ is "feasible" if and only if the corresponding constraints (8-9) are feasible. Then we have:

Proposition 4.2. *For any arbitrary $N \in \mathbb{N}$, if $\overline{\mathcal{T}}_N$ is infeasible, then $\overline{\mathcal{T}}_{N+1}$ is infeasible.*

Proof. By way of contrapositive, we will suppose $\overline{\mathcal{T}}_{N+1}$ is feasible, and then show that $\overline{\mathcal{T}}_N$ must be feasible as well.

Consider any node n in $\overline{\mathcal{T}}_N$ with training stream $\mathcal{D}_n = \langle z_1, z_2, ..., z_n \rangle$, where we set $z_t = x_t \cdot y_t$ to simplify notation. By definition of $\overline{\mathcal{T}}_N$ this stream is linearly separable, meaning there exists some w such that $w^\top z_t > 0$ for each $z_t \in \mathcal{D}_n$. This implies that the $(N+1)$-dimensional vector $[w; 0]$, which is w concatenated with a 0 at the end, also separates the stream

$$\mathcal{D}_{\hat{n}} = \left\langle \begin{bmatrix} z_1 \\ -1 \end{bmatrix}, \begin{bmatrix} z_1 \\ +1 \end{bmatrix}, ..., \begin{bmatrix} z_n \\ -1 \end{bmatrix}, \begin{bmatrix} z_n \\ +1 \end{bmatrix} \right\rangle,$$

where each $[z_t; \pm 1]$ is z_t concatenated with ± 1 at the end. Therefore $\mathcal{D}_{\hat{n}}$ is also linearly separable, and hence there is a node $\hat{n}$ in $\overline{\mathcal{T}}_{N+1}$ with stream $\mathcal{D}_{\hat{n}}$. Likewise, the parent p of n in $\overline{\mathcal{T}}_N$ has a corresponding node $\hat{p}$ in $\overline{\mathcal{T}}_{N+1}$ which is the grandparent of $\hat{n}$, and $\mathcal{D}_{\hat{p}}$ is $\mathcal{D}_{\hat{n}}$ with the last two examples removed.

Having supposed that $\overline{\mathcal{T}}_{N+1}$ is feasible, we have

$$u_{\hat{n}} = u_{\hat{p}} + \gamma_{\hat{n}_-} \begin{bmatrix} z_n \\ -1 \end{bmatrix} + \gamma_{\hat{n}_+} \begin{bmatrix} z_n \\ +1 \end{bmatrix} \tag{20}$$

for some vectors $u_{\hat{n}}$, $u_{\hat{p}}$ and scalars $\gamma_{\hat{n}_-}$, $\gamma_{\hat{n}_+}$, which also satisfy

$$u_{\hat{n}}^\top \begin{bmatrix} z_t \\ -1 \end{bmatrix} > 0 \quad \text{and} \quad u_{\hat{n}}^\top \begin{bmatrix} z_t \\ +1 \end{bmatrix} > 0 \tag{21}$$

for each pair $[z_t; -1]$ and $[z_t; +1]$ in $\mathcal{D}_{\hat{n}}$. Adding the two inequalities in (21), the $\pm$ terms cancel, leaving $\check{u}_{\hat{n}}^\top z_t > 0$, where $\check{u}_{\hat{n}} \in \mathbb{R}^N$ is the vector containing the first N entries of the $(N+1)$-dimensional vector $u_{\hat{n}}$. This means that the vectors $\check{u}_{\hat{n}}$ satisfy constraint (8). Inspecting the first N dimensions of (20) we find that $\check{u}_{\hat{n}} = \check{u}_{\hat{p}} + (\gamma_{\hat{n}_-} + \gamma_{\hat{n}_+}) z_n$, so they also satisfy constraint (9). Therefore, by setting $u_0 = \check{u}_{\hat{0}}$ and $\gamma_n = (\gamma_{\hat{n}_-} + \gamma_{\hat{n}_+})$ for every n, we obtain a feasible solution for $\overline{\mathcal{T}}_N$. □

5. Numerical Methods

Using similar methods to (Winder, 1965; Muroga et al., 1970), we first enumerated all canonical linearly separable dichotomies of H^N up to $N = 9$. These methods also return weight vectors that separate their respective dichotomies. Henceforth, for brevity we will simply write "dichotomies" to mean canonical linearly separable dichotomies, unless otherwise stated. We let X_k denote the k^{th} vertex in H^N, let $Y_{i,k}$ denote the label assigned to X_k by the i^{th} dichotomy, and let W_i denote the returned weight vector for that dichotomy. We arbitrarily chose to index vertices of H^N in lexicographic order (i.e., viewing X_k as the number k expressed in base 2, with 0's replaced by -1's). As an example, X and Y are shown in Figure 3 for $N = 5$.

We then constructed a sub-graph $\hat{\mathcal{T}}$ of $\overline{\mathcal{T}}$ with one leaf per dichotomy, with examples along the path to the leaf

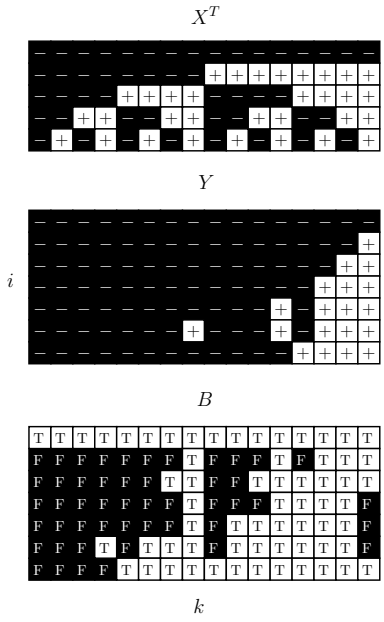

Figure 3. $X^\top$ (top), Y (middle), and B (bottom) for $N = 5$. Indices i and k range over rows and columns, respectively.

presented again in lexicographic order. To reduce $\mathbb{P}(1)$ to a manageable size, we did not include all $x \in H^N$ along the path to the leaf, but only the "boundary" or "support vectors" of the dichotomy corresponding to irredundant constraints. Geometrically, these are the x's whose null-planes form the boundary of the dichotomy's region. For example, in Figure 1 (left), each three-sided region has only three boundary x's, even though there are four x's in H^3. A shown by Fisher & Dearholt (1973), X_k is a boundary vector of region i precisely when there exists another (not necessarily canonical) region j which agrees with i on every input except X_k (i.e., when $Y_{i,k} = -Y_{j,k}$, but $Y_{i,k'} = Y_{j,k'}$ for all other $k' \neq k$). We checked this condition by brute force to form a boolean boundary matrix B, where $B_{i,k}$ is True if X_k is a boundary of region i and False otherwise. Figure 3 (bottom) shows B in the case $N = 5$.

Since our theoretical results depend on the correctness of our code, instead of presenting algorithmic pseudocode, we show our core Python implementation in Listing 1 and explain it here.

First, we check on lines 1-5 that X and Y are ± 1 matrices, the rows of X are unique vertices in H^N, and the rows in Y are unique dichotomies. Next, on lines 9-18, we build the node set V and edge set E of $\hat{\mathcal{T}}$. The outer loop (line 10) creates one leaf per dichotomy, and the inner loop (line 12) creates nodes along the path from root to leaf. Since multiple paths may have overlapping leading portions, care was taken to ensure there were no duplicate nodes where $n \neq n'$ but $\mathcal{D}_n = \mathcal{D}_{n'}$. This was done by implementing V as a key-value lookup table, using each unique $\mathcal{D}_n$ as a key that maps to the corresponding node index n as its value (along with current dichotomy i, for later use). $\mathcal{D}_0$ is initialized empty on line 11 and examples are appended to it on line 14 as the path is traversed. We used the index k rather than input vector X_k in the key, since the former could be stored directly as a key in a Python dictionary. New nodes and edges were only allocated on lines 16-18 if they were not duplicates of previously constructed leading paths, which was checked by the if-statement on line 15. Finally, lines 20-24 reconstruct the input vectors X_k in each $\mathcal{D}_n$ from their k's in the corresponding key.

```
1  N = X.shape[1]
2  assert (np.fabs(X) == 1).all()
3  assert (np.fabs(Y) == 1).all()
4  assert len(X) == len(np.unique(X, axis=0))
5  assert len(Y) == len(np.unique(Y, axis=0))
6
7  print("Building the tree...")
8
9  V, E = {(): (0, 0)}, []
10 for i in range(len(Y)):
11   D = ()
12   for k in np.flatnonzero(B[i]):
13     p, _ = V[D]
14     D += ((k, Y[i,k]),)
15     if D not in V:
16       n = len(V)
17       V[D] = (n, i)
18       E.append( (n, p, X[k], Y[i,k]) )
19
20 D = [(X[[]], np.empty(0), 0)]
21 for (Dn, (n, i)) in V.items():
22   if n == 0: continue
23   ks, y = map(np.int64, zip(*Dn))
24   D.append((X[ks], y, i))
25
26 print("Checking the tree...")
27
28 for (Xn, yn, i) in D:
29   assert (np.sign(W[i] @ Xn.T) == yn).all()
30 for (n, p, x, y) in E:
31   Xn, yn, _ = D[n]
32   Xp, yp, _ = D[p]
33   assert (Xp == Xn[:-1]).all()
34   assert (yp == yn[:-1]).all()
35   assert (x == Xn[-1]).all()
36   assert (y == yn[-1]).all()
37
38 print("Running the linear program...")
39
40 u = cp.Variable((len(D), N))
41 g = cp.Variable(len(E))
42
43 span_constraints = [
44   u[n] == (u[p] + g[e] * x)
45   for e, (n, p, x, _) in enumerate(E)]
46
47 data_constraints = [
48   u[n] @ (Xn.T * yn) >= 1
49   for n, (Xn, yn, _) in enumerate(D) if n > 0]
50
51 c = np.stack([
52   (Xn.T * yn).sum(axis=1)
53   for n, (Xn, yn, _) in enumerate(D) if n > 0])
54
55 constraints = span_constraints + data_constraints
56 objective = cp.Minimize(cp.sum(cp.multiply(u[1:],c)))
57
58 problem = cp.Problem(objective, constraints)
59 problem.solve(solver=solver, verbose=verbose)
```

Listing 1. Tree construction and feasibility check

Since the correctness of the code on lines 9-24 may not be obvious, we perform additional checks that the constructed tree is indeed a sub-graph of $\overline{\mathcal{T}}$. The assertions on lines 28-29 show that each $\mathcal{D}_n$ is indeed linearly separable. Here we made use of the separating vector W_i which was previously identified during region enumeration, using the methods of Winder (1965) and Muroga et al. (1970). This check was the sole purpose of saving each i along with node index n earlier on line 17; after this point i and W_i are not used.

The second set of assertions on lines 30-36 confirm that each node's data stream $\mathcal{D}_n$ is indeed equivalent to its parent's data stream $\mathcal{D}_{p(n)}$, with the one new training example (x_n, y_n) for the current edge appended at the end.

Finally, lines 40-59 encode and solve $\mathbb{P}(1)$ using CVXPY (Diamond & Boyd, 2016; Agrawal et al., 2018). We compared several commercial and open-source solvers, including Gurobi, MOSEK, CBC (Lougee-Heimer, 2003), ECOS (Domahidi et al., 2013), and the HiGHS solver in SciPy (Huangfu & Hall, 2018; Virtanen et al., 2020).

For $N = 8$ where full-capacity learning turned out to be infeasible, we also gauged the extent to which "high" (but not full) capacity learning was possible using Monte-Carlo experiments. Specifically, we randomly sampled a subset of the dichotomies and constructed $\hat{\mathcal{T}}$ using only that subset. In other words, we randomly discarded many rows of Y and B before passing them to the code in Listing 1. The subset size was an experimental parameter that we varied between 2 and 2470 (the latter is the total number of dichotomies for $N = 8$). The sizes we checked were spaced equally across that range (excluding 2470, which was already found infeasible in the full-capacity check). To quantify capacity we define the "feasibility rate" at each subset size as the fraction of 30 independent random repetitions where $\mathbb{P}(1)$ was feasible. Only Gurobi was used here because it was one of the fastest and most reliable solvers on our problem.

Since several solvers had occassional numerical issues, we deemed it necessary to also check an infeasible sub-graph by hand and certify that it was indeed infeasible. This check is covered in Section 7. For this purpose, it was important to identify a very small infeasible sub-graph, which we did as follows. First, we sorted the dichotomies by their number of boundary vectors in ascending order, since fewer boundary vectors tend to produce smaller sub-trees. Next, we iterated over all size-2 subsets of dichotomies according to this order, so that pairs whose dichotomies had fewer boundary vectors would be checked earlier in the iteration. We checked pairs rather than singletons because individual dichotomies are always feasible, due to their linear separability and the span coefficients guaranteed by the representer theorem. It turned out that an infeasible pair of dichotomies did exist. We terminated the search once this pair was found numerically, and certify its infeasibility by hand in Section 7.

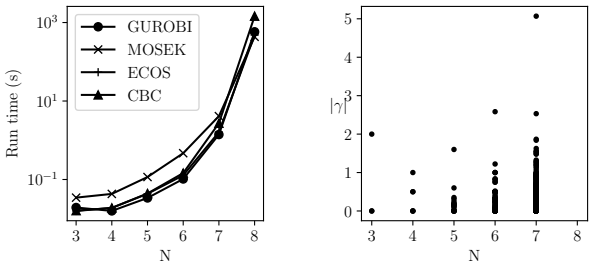

Figure 4. **Left:** Running times to solve $\mathbb{P}(1)$ for $3 \leq N \leq 8$. **Right:** Numerical values of $|\gamma_n|$, one dot per node n in $\hat{\mathcal{T}}$, at the optimal solutions found by Gurobi ($N = 8$ was infeasible).

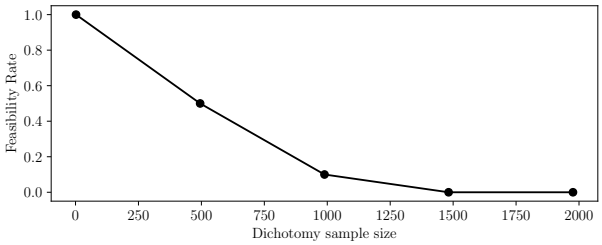

Figure 5. For $N = 8$, fraction of repetitions ("feasibility rate") where $\mathbb{P}(1)$ was reported feasible by Gurobi, when constructing $\hat{\mathcal{T}}$ using a random subset of canonical dichotomies. Subset sizes were equally spaced at 5 points between 2 and the total number of canonical dichotomies. For each subset size, 30 repetitions were used to estimate the feasibility rate.

All experiments were done on a workstation with 8-core Intel i7 CPU and 32GB of RAM, Fedora 39 Linux, Python 3.11.7, NumPy 1.26.3, SciPy 1.11.1, CVXPY 1.4.1, Gurobi 11.0.0, and MOSEK 10.1. Run-times depended on N and the solver used, but for Gurobi the full set of experiments completed in roughly two days. All experiment code is open-source (MIT license) and freely available online.[1]

6. Numerical Results

Figure 4 (left) shows the running times for various solvers on each input dimension tested. While run times varied, all solvers that terminated agreed that $\mathbb{P}(1)$ was feasible for $N \leq 7$ and infeasible for $N = 8$. The HiGHS solver ran indefinitely at $N = 8$ so we exclude it from the figure. Numerical values of $|\gamma_n|$ at the feasible solutions found by Gurobi for $N \leq 7$ are shown in Figure 4 (right). We confirmed separately that, using these numerical solutions for $N \leq 7$, the maximum residual error in any constraint was near machine precision.

[1] https://github.com/garrettkatz/slim

Figure 5 shows the results of our high-capacity experiments, using random sub-samples of dichotomies in $\mathbb{P}(1)$. The results in Figure 5 are consistent with the existence of an infeasible size-2 subset. For example, the probability that a random half of the dichotomies includes a particular size-2 subset is roughly 0.25, and we observe a feasibility rate less than 0.25 when roughly half of the 2470 dichotomies are in the sample. In fact, the feasibility rate for a random half is closer to 0.1, suggesting the existence of other infeasible subsets in addition to the one that we found.

That said, it was not practical to double-check infeasibility of each sample "by hand" since there are 30×5 samples and most are very large. Therefore, the results in Fig. 5 are suggestive but not conclusive, given the occassional numerical mistakes made by the solvers. On the other hand, we do hand-check one infeasible size-2 subset in the following section, so our main results in Section 4 stand.

7. Infeasibility Certificate

In principle, linear programming is a convex optimization problem that can be solved with strong formal guarantees. For example, the simplex method is guaranteed to terminate under appropriate pivoting rules that account for degeneracy (Bertsimas & Tsitsiklis, 1997). Similar guarantees are known for interior point methods. However, in practice, modern solvers may implement more complex variants of these approaches and are not always open source; so it is difficult to determine precise guarantees for the specific solvers we used. Numerical issues in these solvers, though rare, did occur. Therefore our theoretical results are not fully established unless we prove "by hand" that $\mathbb{P}(1)$ truly is infeasible, for at least one sub-graph $\hat{\mathcal{T}}$ of $\overline{\mathcal{T}}$. Although HiGHS was the only solver that ran indefinitely, we found at various stages of the process (region enumeration, boundary identification, full and high capacity checks) that ECOS occassionally produced false positives (reported an infeasible program as "feasible"), and CBC occasionally produced false negatives (reported a feasible program as "infeasible"). We were able to detect these issues when the outputs of two solvers disagreed. MOSEK occasionally reported that it could not solve certain instances.

Gurobi was the only solver for which we did not detect any inconsistencies. However, given the foregoing issues, it was still important to certify the infeasibility claims by hand. In this section we present an infeasibility certificate for a specific infeasible pair of dichotomies at $N = 8$: namely, the pair whose sub-graph was shown previously in Figure 2.

Specifically, we will show that the sub-graph rooted at node 6 in Figure 2 is infeasible. To simplify the notation, let Z_n denote the matrix whose columns are the products xy for each $(x, y) \in \mathcal{D}_n$. Next, to reduce $\mathbb{P}(1)$ to a reasonable size for manual analysis, we eliminate the variables u_n for $n > 6$ by rewriting them in terms of u_6 and the γ_n's, according to condition (9), which gives the system:

$$
\begin{aligned}
&(u_6 && &&)^\top Z_6 \geq 1\\
&(u_6 + \gamma_7 x_7 && &&)^\top Z_7 \geq 1\\
&(u_6 + \gamma_7 x_7 + \gamma_8 x_8 && &&)^\top Z_8 \geq 1\\
&(u_6 && + \gamma_9 x_9 &&)^\top Z_9 \geq 1\\
&(u_6 && + \gamma_9 x_9 + \gamma_{10} x_{10} &&)^\top Z_{10} \geq 1\\
&(u_6 && + \gamma_9 x_9 + \gamma_{10} x_{10} + \gamma_{11} x_{11} &&)^\top Z_{11} \geq 1
\end{aligned}
$$

Each row of this system corresponds to condition (8) at one node in the sub-graph. We can reorganize this system into the form $Av \geq \mathbf{1}$, where $\mathbf{1}$ is a vector of all 1's and v is the concatenation of u_6 with γ_7 through γ_{11}. In this form, A is a block matrix whose blocks include copies of $Z_n^\top$ and various products $Z_n^\top x_m$ for certain node pairs (n, m), such as $Z_8^\top x_7$ and $Z_8^\top x_8$ from the third inequality in the system. The full matrix A is shown in Figure 6.

It remains to show that the system of inequalities $Av \geq \mathbf{1}$ is infeasible. If there existed a v satisfying this system, then it would also satisfy $s^\top Av \geq s^\top \mathbf{1}$ for any vector s with non-negative entries, since a non-negative scaling of any inequality does not change the direction of the inequality. However, in Figure 6 we exhibit one such s for which $s^\top A = \mathbf{0}$, where $\mathbf{0}$ is a vector of all 0's, but $s^\top \mathbf{1} = 20$. This would imply $s^\top Av = \mathbf{0}$ and hence $0 \geq 20$, a contradiction. Therefore, no v satisfying $Av \geq \mathbf{1}$ can exist, certifying that $\mathbb{P}(1)$ is indeed infeasible on this particular sub-graph of $\overline{\mathcal{T}}$.

8. Discussion and Limitations

Our results help characterize the theoretical limits of linear threshold learning with binary input vectors. In particular, any learning process taking the form of a span rule can not be both single-pass and full-capacity. The upshot is that this result imposes a design constraint on single-pass learning processes. When researching new methods for single-pass full-capacity learning, candidates that take the form of span rules can be removed from consideration to avoid wasted search effort. This applies to manual analyses as well as automated search methods, e.g., symbolic regression (La Cava et al., 2021). Automated search for learning rules is a challenging problem, but some inroads have been made in recent years, such as (Lindsey & Litwin-Kumar, 2020).

One limitation of our work is that many real-world datasets are not binary. Strictly speaking, since $H^N \subseteq \mathbb{R}^N$, we have also provided a counter-example to single-pass, full-capacity span learning in the case of real-valued input. However, these counterexamples might be rare when considering typical statistical distributions over real-valued training data, such as vectors in general position, or the highly non-

uniform distribution of natural images. One avenue for future work is therefore extending our approach to real-valued input vectors from various statistical distributions.

Another limitation is our focus on learning in a single linear threshold unit, as opposed to deep multi-layer networks of such units, or other architectures such as attention-based models. The single-unit case was a natural starting point, since it already proved to be fairly complicated, and to our knowledge even in this case the feasibility question had not been previously answered. That said, it is well-known that multi-layer networks have higher cardinal capacity and can fit non-linearly-separable data (Baldi & Vershynin, 2019). So another avenue for future work is to investigate the feasibility of single-pass span rule learning in a multi-layer context, including the overparameterized regime, and incorporate recent work in that area such as (Min et al., 2022; Zhu & Xu, 2021). Extending our methods to multi-layer networks appears challenging, due to the hidden layer non-linearities. However, there is some recent progress on using numerical techniques to interpret multi-layer networks with real-valued (ReLU) activations, such as (Rolnick & Kording, 2020). In our case we believe something similar may be possible, although we will most likely have to resort to mixed-integer or otherwise non-linear optimization.

We also note several real-world applications for binary input and/or single-layer networks. For example, discrete as opposed to continuous input is relevant to quantized models on low-memory edge devices (Hubara et al., 2018). Even when memory is not constrained, discrete representation learning (i.e., VQ-VAE (Van Den Oord et al., 2017)) has certain benefits and is a widely-used technique. Single-layer linear associative networks also have utility in certain settings, such as neural program induction architectures, in which they are used to emulate random-access computer memory (Katz et al., 2020).

Lastly, future work should address not only the model's data-fitting capacity, but also its generalization performance. One possible approach is to augment the linear constraints with margin-maximization objectives. In constructing the sub-graph $\hat{\mathcal{T}}$, we only included irredundant constraints, i.e., the "support vectors" of each dichotomy, which are relevant to efficient margin maximization. So there is some basis to believe our methodology could be extended in this direction.

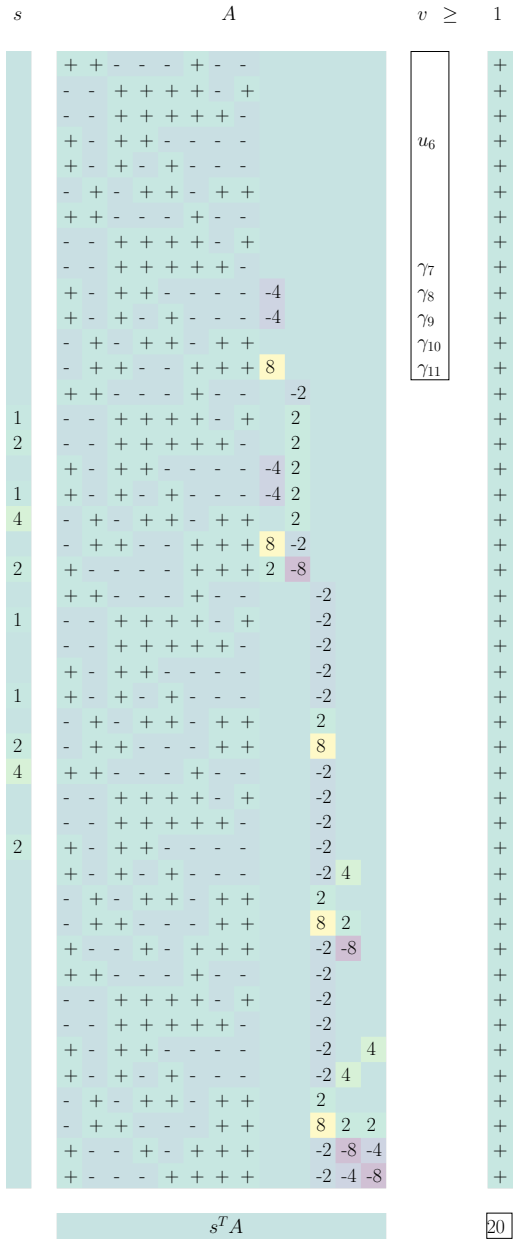

Figure 6. Certificate that $Av \geq \mathbf{1}$ is infeasible. The large matrix in the center is A; the short column vector to its right shows the arrangement of u_6 and the γ_n's in v. The column vectors on far left and right are s and $\mathbf{1}$. The row vector at bottom center is $s^\top A$, and the scalar at bottom right is $s^\top \mathbf{1}$. To reduce clutter, zero entries are unlabeled, and for ± 1 entries only the sign is displayed.

Acknowledgements

This work is partially supported by the National Science Foundation I/UCRC ASIC (Alternative Sustainable and Intelligent Computing) Center (CNS-1822165), and the Center for Advanced Systems and Engineering at Syracuse University, a New York State Center for Advanced Technology. Thanks also to the reviewers for their helpful feedback.

Impact Statement

This work makes a small contribution to the field of single-pass machine learning, and more generally, efficient machine learning. Reducing the computational costs of machine learning can have positive impacts on the environment as well as under-resourced researchers and practitioners of AI. On the other hand, faster learning can also further accelerate the advancement of AI towards human-level intelligence and beyond, which has numerous associated risks.

References

Agrawal, A., Verschueren, R., Diamond, S., and Boyd, S. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

Alon, N., Matias, Y., and Szegedy, M. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 20–29, 1996.

Amit, D. J., Gutfreund, H., and Sompolinsky, H. Storing infinite numbers of patterns in a spin-glass model of neural networks. *Physical Review Letters*, 55(14):1530, 1985.

Anderson, J. A. A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14(3-4): 197–220, 1972.

Baldi, P. and Vershynin, R. The capacity of feedforward neural networks. *Neural Networks*, 116:288–311, 2019.

Bertsimas, D. and Tsitsiklis, J. N. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

Block, H.-D. The perceptron: A model for brain functioning. I. *Reviews of Modern Physics*, 34(1):123, 1962.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152, 1992.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Voss, A. H., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

Carvalho, V. R. and Cohen, W. W. Single-pass online learning: Performance, voting schemes and online feature selection. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 548–553, 2006.

Charikar, M., O'Callaghan, L., and Panigrahy, R. Better streaming algorithms for clustering problems. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pp. 30–39, 2003.

Chow, C.-K. On the characterization of threshold functions. In *2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1961)*, pp. 34–38. IEEE, 1961.

Cover, T. M. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, (3):326–334, 1965.

Demircigil, M., Heusel, J., Löwe, M., Upgang, S., and Vermet, F. On a model of associative memory with huge storage capacity. *Journal of Statistical Physics*, 168(2): 288–299, 2017.

Diamond, S. and Boyd, S. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Domahidi, A., Chu, E., and Boyd, S. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)*, pp. 3071–3076, 2013.

Domingos, P. and Hulten, G. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 71–80, 2000.

Elizondo, D. The linear separability problem: Some testing methods. *IEEE Transactions on Neural Networks*, 17(2): 330–344, 2006.

Fei-Fei, L., Fergus, R., and Perona, P. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, 2006.

Fisher, L. T. and Dearholt, D. W. Boundary points of threshold functions. *IEEE Transactions on Computers*, 100(12): 1132–1139, 1973.

Gardner, E. The space of interactions in neural network models. *Journal of physics A: Mathematical and general*, 21(1):257, 1988.

Goto, E. and Takahasi, H. Some theorems useful in threshold logic for enumerating boolean functions. In *IFIP congress*, pp. 747–752, 1962.

Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

Huangfu, Q. and Hall, J. J. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18(187):1–30, 2018.

Katz, G. E., Gupta, K., and Reggia, J. A. Reinforcement-based program induction in a neural virtual machine. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2020.

Kohonen, T. Correlation matrix memories. *IEEE Transactions on Computers*, 100(4):353–359, 1972.

Kohonen, T. *Self-organization and associative memory*, volume 8. Springer Science & Business Media, 2012.

Krotov, D. and Hopfield, J. J. Dense associative memory for pattern recognition. *Advances in Neural Information Processing Systems*, 29, 2016.

La Cava, W., Orzechowski, P., Burlacu, B., de Franca, F. O., Virgolin, M., Jin, Y., Kommenda, M., and Moore, J. H. Contemporary symbolic regression methods and their relative performance. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Li, Y. and Long, P. The relaxed online maximum margin algorithm. *Advances in neural information processing systems*, 12, 1999.

Lindsey, J. and Litwin-Kumar, A. Learning to learn with feedback and local plasticity. *Advances in Neural Information Processing Systems*, 33:21213–21223, 2020.

Linnainmaa, S. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. PhD thesis, Master's Thesis (in Finnish), Univ. Helsinki, 1970.

Liu, Y. and Xu, J. One-pass online SVM with extremely small space complexity. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 3482–3487. IEEE, 2016.

Lougee-Heimer, R. The common optimization interface for operations research: Promoting open-source software in the operations research community. *IBM Journal of Research and Development*, 47(1):57–66, 2003.

McGregor, A. Graph stream algorithms: a survey. *ACM SIGMOD Record*, 43(1):9–20, 2014.

Min, Y., Ahn, K., and Azizan, N. One-pass learning via bridging orthogonal gradient descent and recursive least-squares. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pp. 4720–4725. IEEE, 2022.

Muroga, S., Tsuboi, T., and Baugh, C. R. Enumeration of threshold functions of eight variables. *IEEE Transactions on Computers*, 100(9):818–825, 1970.

Murphy, C., Gray, P., and Stewart, G. Verified perceptron convergence theorem. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pp. 43–50, 2017.

Ojha, P. C. Enumeration of linear threshold functions from the lattice of hyperplane intersections. *IEEE Transactions on Neural Networks*, 11(4):839–850, 2000.

Picton, P. Threshold logic: Is there finally a solution? In *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 45–51. IEEE, 2016.

Rai, P., Daumé, H., and Venkatasubramanian, S. Streamed learning: one-pass SVMs. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pp. 1211–1216, 2009.

Rao, Y. and Zhang, X. The characterizations of hyper-star graphs induced by linearly separable boolean functions. *Chinese Journal of Electronics*, 27(1):19–25, 2018.

Rolnick, D. and Kording, K. Reverse-engineering deep relu networks. In *International conference on machine learning*, pp. 8178–8187. PMLR, 2020.

Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Schölkopf, B., Herbrich, R., and Smola, A. J. A generalized representer theorem. In *International conference on computational learning theory*, pp. 416–426. Springer, 2001.

Slepian, D. On the number of symmetry types of boolean functions of n variables. *Canadian Journal of Mathematics*, 5:185–193, 1953.

Todd, J. The groups of symmetries of the regular polytopes. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 27, pp. 212–231. Cambridge University Press, 1931.

Van Den Oord, A., Vinyals, O., et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. Matching networks for one shot learning. *Advances in Neural Information Processing Systems*, 29, 2016.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Widrow, B. and Hoff, M. E. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.

Winder, R. O. Enumeration of seven-argument threshold functions. *IEEE Transactions on Electronic Computers*, (3):315–325, 1965.

Winder, R. O. Partitions of n-space by hyperplanes. *SIAM Journal on Applied Mathematics*, 14(4):811–818, 1966.

Zhu, H. and Xu, J. One-pass stochastic gradient descent in overparametrized two-layer neural networks. In *International Conference on Artificial Intelligence and Statistics*, pp. 3673–3681. PMLR, 2021.

Zuev, Y. A. Asymptotics of the logarithm of the number of threshold functions of the algebra of logic. In *Soviet Math Dokl*, volume 39, pp. 512–513, 1989.