A strongly dual approach for building interpretable binary activated neural networks

Benjamin Leblanc
Université Laval
Canada
benjamin.leblanc.2@ulaval.ca

1 Introduction

In recent years, neural networks (NNs) has overtaken state of the art in several domains, namely in natural language processing [1] and computer vision [2, 3]. These advances allow one with important computational and memory resources to train truly deep and wide neural networks on enormous corpus of text or image datasets containing millions of examples. It is a widespread thought that the predictive performance of a NN increase with its number of parameters, motivating researchers to always train bigger and bigger NNs [4–7]. The main drawback of these models is that their huge complexity prohibit their deployment in scenarios where being able to interpret the predictor is of importance. Various techniques have been proposed to lighten NNs in order to make them simpler; in addition to being natural regularizer [8, 9], the use of weights encoded by a small number of bits [10–12] and quantized activation functions, notably binary activation functions [8, 13, 14], have shown promising results. Also, methods such as network pruning [15–19], weight sharing [20] or matrix factorisation [21] allow compacting a trained NN, resulting in lighter predictors.

The aforementioned processes involve many manipulations : first, it is necessary to train several deep and wide NNs (with various combinations of hyperparameters and architectures). Then, they must be pruned using, once again, various techniques, having no guarantee that trained network that obtain the best performances will still be the best pruned network. Plus, as implied by the Lottery Ticket Hypothesis from Frankle and Carbin [22], in order to obtain an efficient compact network from a pruned bigger network, the latter must be big enough so that the chances are high that, within a few different random seeds, it contains a compact but performing sub-network. Thus, it seems like a lot of work for obtaining, in the end, a small and compact predictor. Working backward would be an interesting avenue in order to obtain such networks without needing important computational resources to train wide NNs first; starting with small NNs and making them grow, just like a decision tree is built. Unfortunately, while it's been shown that greedy approaches can scale to colossal problems such as ImageNet [23], the literature concerning greedy approach for training NNs is pretty arid [24–26], and even more for networks with quantized components.

We propose CPNet, an algorithm for tackling binary classification tasks and greedily building compact and interpretable binary activated neural networks (BANNs), one neuron at a time, with the help of constraint programming. It is the first time, to the knowlodge of the authors, that binary activated neural networks are both studied as interpretable predictors and trained with CP by making use of the fact that each neuron of such networks can be seen as a separating hyperplane. CPNet uses a dual optimization scheme; we propose two distinct way to obtain an optimal 0-1 loss when manipulating a separating hyperplane. Finally, we present an interesting result giving a condition for a dataset to be linearly separable.

2 Related work

Constraint programming (CP) and mixed-integer linear programming (MILP) While some of the literature involving both CP or MILP and binary neural networks concerns the filed of adversarial

attack [27], some tackles the problem of neural network verification [28–30]. Binary neural networks have been trained [31] or optimized [32] with CP, but mostly by considering not only the activations to be binary, but the weights as well.

Greedy neural network training Greedy approaches for training NNs usually refers to the use of a layer-wise training scheme [23–25, 33–35]. The usual procedure is the following: first, the full network is trained. Then, the first layer of the network is fixed. The network is retrained, where each layer but the first is fine-tuned. The second layer of the network is then fixed, and so on. Only a few paper consider building one neuron at a time the network [35].

3 Details on the problem

Interpretable BANNs We'll use the Rudin's paradigm of interpretability [36, 37] which roughly goes as follow: a predictor is interpretable if one can understand and comprehend its decision process solely by considering the predictor in itself. Such an approach thus prohibit cases where explainability techniques such as LIME [38] or the SHAP values [39] are used to understand the predictor.

Let's take a look at what an interpretable BANN might look like. Let's consider the task of predicting the costs of insuring a house; let a be its age, b be the number of floors it has, B a BANN and $\mathbb{1}\{\cdot\}$ the indicator function. Then, B could be defined as follow:

$$B(a, b) = 50 + 100 * \mathbb{1}\{a > 50\} + 200 * \mathbb{1}\{b > 1\}.$$

Simply by looking at it, we understand that the base cost for insuring a house, in this case, is 50\$. Then, if the house is more then 50 years old, is costs 100\$ more, probably because older houses are more prone to be problematic. Finally, if the house has more than 1 floor, it costs 200\$ more, probably because the number of floor of a house has a proportional relationship with it's value. Overall, the role of each parameter of the model is easily understandable.

We'll focus on predictors where the indicator function 1. do not contain more than 2 variables (more than 2 might be a bit to complex to easily interpret) and 2. do not contain any indicator function. Basically: our predictor will be 1. single-hidden layered and 2. each hidden neuron will be parameterized by at most two non-zero weights.

Tasks From now on, we'll be focusing on binary classification tasks.

A binary neuron is a separating hyperplane An important thing to understand is that a hidden neuron, as well as the output neuron of any binary classifier, can be seen as a separating hyperplane. Indeed, those neurons acts as follow: $N(\mathbf{x}; \mathbf{w}, b) = \mathbb{1}\{\langle \mathbf{x}, \mathbf{w} \rangle + b > 0\}$. Thus, $\mathbf{w}$ corresponds to the orientation of a separating hyperplane, while b corresponds to its bias and $N(\mathbf{x}; \mathbf{w}, b)$ tells whether $\mathbf{x}$ is on one side of the hyperplane or the other.

SVM If the goal, when parameterizing a hidden neuron, is thus to find the best separating hyperplane, why not simply tackle this problem using a soft-margin SVM, with a linear kernel and a truly small regularisation parameter? Well, this approach minimizes the L2 norm of the orientation of the hyperplane and the obtained *hinge* loss, a soft surrogate for the 0-1 loss. Therefore, even though it is sure to converge, it won't lead to the best classifier from the 0-1 loss point of view. We've got to tackle this problem without using a surrogate loss function in order to obtain compact predictors.

The method Since we build our network greedily, we alternate between adding (and parameterizing) a new neuron on the hidden layer and updating the weights and bias of the ouput layer (see Figure 2 in appendix for a visual representation of the building steps). We'll be using threshold activation function instead of the more common sign function for interpretability purposes (it's been showed that the parameterization of the binay activation function has no impact on the space of obtainable predictor [35]).

4 One problem, two perspectives

At each step of the algorithm, it all comes to this: finding the best separating hyperplane considering a set of examples being labeled 1 (positive) and a set of examples being labeld 0 (null). For generality

sake, we will considered weighted examples, so that misclassifying a certain example might be less important than misclassfying others. Let's look at two approaches for tackling our problem.

4.1 Primal

The primal approach is the most natural way to tackle the problem; we try to place the hyperplane in the feature space such that it leads to the best separation of the examples (best 0-1 loss). Therefore, it is a maximization problem; we try to maximize this accuracy. This approach also leads to a pretty simple CP model:

Primal model explaination

- n_1 : number of example of the first class;
- n_2 : number of example of the second class;
- d : dimension of the examples;
- c : maximum number of non-zero values for **w**;
- A : $n_a \times d$ matrix containing every first class points;
- B : $n_b \times d$ matrix containing every first class points;
- *ValA* : vector of lenght n_a containing the weights of each datapoint of the first class (f.c.);
- *ValB* : vector of lenght n_b containing the weights of each datapoint of the second class (s.c.);
- Variables $w_i \forall i \in \{1, \ldots, d\}$: correspond to the orientation of the separating hyperplane to be placed;
- Variable *b* : correspond to the bias of the separating hyperplane to be placed;
- Variables $ind_i \in \{0, 1\} \forall i \in \{1, \ldots, d\}$: correspond to an indicator telling whether w_i is 0 or not $\forall i \in \{1, \ldots, d\}$.
- Constraint #1 : makes sure that the value of ind_i is 1 if w_i is different from 0, and 0 otherwise $\forall i \in \{1, \ldots, d\}$ (there are d of them);
- Constraint #2 : makes sure that the L1 norm of the vector w is equal to one (there is a single one);
- Constraint #3 : makes sure m corresponds to the minimal value of a data point to the hyperplane; (there is a single one);
- Constraint #4 : makes sure that at least $d + 1$ points share the minimal distance from a point to the hyperplane (there is a single one);
- Constraint #5 : makes sure at most p component of **w** are different from 0 (there is a single one).

Primal MiniZinc model (1/2)

n_1	Number of data points - f.c.
n_2	Number of data points - s.c.
m	Minimal distance of a point to the hyperplane
d	Dimension of the data points
c	Maximum number of non-zero values for **w**
$A \in [0, 1]^{n_a}$	Data points - f.c.
$B \in [0, 1]^{n_b}$	Data points - s.c.
$ValA \in [0, 1]^d$	Weights - f.c.
$ValB \in [0, 1]^d$	Weights - s.c.
$\text{dom}(w_i) = [-1, 1]$	$\forall i \in \{1, \ldots, d\}$
$\text{dom}(b) = [-d, d]$	

Primal MiniZinc model (2/2)

$$\text{dom}(ind_i) = \{0, 1\} \qquad \forall i \in \{1, \dots, d\}$$

$$\sum_{i \in \{1,\dots,n_a\}} ValA_i * \mathbb{1}\{\langle \mathbf{w}, A_i \rangle + b \geq 0\} + \sum_{i \in \{1,\dots,n_b\}} ValB_i * \mathbb{1}\{\langle \mathbf{w}, B_i \rangle + b < 0\} \qquad \text{Objective function}$$

$$(w_j \neq 0 \wedge ind_j = 1) \vee (w_j = 0 \wedge ind_j = 0) \qquad \forall j \in \{1, \dots, d\} \quad (1)$$

$$\sum_{j \in \{1,\dots,d\}} |w_j| = 1 \quad (2)$$

$$m = min\left(\min_{i \in \{1,\dots,n_a\}}\left(|\langle \mathbf{w}, A_i \rangle + b|\right), \min_{i \in \{1,\dots,n_b\}}\left(|\langle \mathbf{w}, B_i \rangle + b|\right)\right) \quad (3)$$

$$\sum_{i \in \{1,\dots,n_a\}} \mathbb{1}\{|\langle \mathbf{w}, A_i \rangle + b| = m\} + \sum_{i \in \{1,\dots,n_b\}} \mathbb{1}\{|\langle \mathbf{w}, B_i \rangle + b| = m\} > d \quad (4)$$

$$\sum_{j \in \{1,\dots,d\}} ind_j \leq c \quad (5)$$

In order to make the training phase simpler for the solver, we consider each of the examples having values in [0,1] for each of their dimensions (the input have been *min-max* normalized). Since $\text{sign}(x) = \text{sign}\left(\frac{x}{c}\right) \forall c \in \mathbb{R}_+^*$, we can consider without loss of generality that the vector parameterizing the hyperplane to have values in [-1,1]. As for the bias, since the hyperplane has values in [-1,1] and the examples have values in [0,1], than the biases needs not to be in a bigger range than [-d,d] since, for any example $\mathbf{x}$, $|\langle \mathbf{x}, \mathbf{w} \rangle| \leq d$. Doing so leads to considering closed ranges for $\mathbf{w}$ and b during the training.

We also considered two tricks in order to make the solver find the optimal solution faster, considering that an infinite number of hyperplanes can lead to the same classifications:

1. We tried fixing the L1 norm of the variables $\mathbf{w}$. The idea is that the scale of the parameters has no effect on the prediction, so their norm can be fixed without impacting the obtained solution space (Constraint #2). We used the L1 norm so that a solver only able to manipulate single order constraints is able to manipulate that one.
2. We demanded having at least $d + 1$ points sharing the smallest distance to the hyperplane. This can be done considering that for any separation made by a hyperplane in d dimension, there exists such a configuration (Constraint #3-#4).

In the end, both tricks did not lead to faster training of the solver, so we removed them. It seems that the computation cost of such constraints hinder the computational gains of avoiding redundant solutions.

4.2 Dual

The dual approach of our problem emerges from the following consideration: a hyperplane separating the example space probably makes some misclassifications; but, when those misclassified examples are removed from the dataset, than the remaining points are linearly separable. Therefore, our problem can be seen as follow: what is the set of points having the minimal cumulative weight that can be "removed" (unconsidered) from our dataset such that the remaining points are linearly separable?

Now, we need a way to effeciently compute which points contribute to the non-linear separability of the dataset. Our approach will be based on the following (original) result:

Lemma 1 (Linear separability criteria)**.**
Let $\mathbf{A} \in \mathbb{R}^{d \times n_A} \times \{1\}^{n_A}$, $\mathbf{B} \in \mathbb{R}^{d \times n_B} \times \{1\}^{n_B}$ *be two sets of points from* $\mathbb{R}^d \times \{1\}$. *Let* $LS(\mathbf{A}, \mathbf{B})$ *be true if* $\mathbf{A}$ *and* $\mathbf{B}$ *are linearly separable, and false otherwise. Then:*

$$LS(A, B) \Leftrightarrow (\nexists \mathbf{c} \in \mathbb{R}^{n_A}, \mathbf{d} \in \mathbb{R}^{n_B} \mid \mathbf{Ac} = \mathbf{Bd} \neq \mathbf{0}) .$$

The proof of Lemma 1 can be found in appendix. What that last result tells us it that if sets of points $\mathbf{A}$ and $\mathbf{B}$ that aren't linearly separable, then there exists a linear combination of some points from $\mathbf{A}$ being equal to a linear combination of some points from $\mathbf{B}$. Finding such linear combinations gives us insights on which subsets of points contribute to the non-linear separability of our problems.

Our algorithm will thus have the following behaviour, involving two solving models that interact with each other: model #1 tries to find a subset of points from A and B that is not linearly separable and feeds it to the model #2, determining which weighted point should be neglected so that the remaining ones become linearly separable. The solution (let's say $\mathbf{C}$) is fed to model #1, now trying to find a subset of non-linearly separable points that does not involve any points from $\mathbf{C}$ and that is different from the solution that it has previously found. The constraint is fed to model #2, now trying to find which points should be neglected, considering two sets of points (possibly not disjoint) that aren't linearly separable. This goes on until model #1 can't find any non-linearly separable subsets, meaning the optimal separation is found

Dual model #1 explaination (1/2)

- n_1 : number of example of the first class;
- n_2 : number of example of the second class;
- d : dimension of the examples;
- pA : binary vector of length n_a telling which datapoints of the first class not to include in the linear combination;
- pB : binary vector of length n_b telling which datapoints of the second class not to include in the linear combination;
- A : $n_a \times d$ matrix containing every first class points;
- B : $n_b \times d$ matrix containing every first class points;
- *ValA* : vector of lenght n_a containing the weights of each datapoint of the first class;
- *ValB* : vector of lenght n_b containing the weights of each datapoint of the second class;
- Variables $WhiA_i \forall i \in \{1, \ldots, n_a\}$: corresponds to the linear combination coefficient of first class points;
- Variables $WhiB_i \forall i \in \{1, \ldots, n_b\}$: corresponds to the linear combination coefficient of second class points;
- Variables $IndA_i \forall i \in \{1, \ldots, n_a\}$: tells whether or not $WhiP_i$ is 0 or not;
- Variables $IndB_i \forall i \in \{1, \ldots, n_b\}$: tells whether or not $WhiM_i$ is 0 or not;
- Variables $TriA_i \forall i \in \{1, \ldots, n_a\}$: contains the weight value of a data point if that point is included in the linear combination, and the maximum weight value (1) otherwise; is useful for rendering a set of points for which the minimal weight value is maximized;
- Variables $TriB_i \forall i \in \{1, \ldots, n_b\}$: contains the weight value of a data point if that point is included in the linear combination, and the maximum weight value (1) otherwise;
- Constraint #1 : makes sure that a linear combination of A is found so that it equals a linear combination of B (there is a single one);
- Constraint #2 : makes sure that $IndA_i$ reflects whether data point A_i is included in the linear combination or not $\forall i \in \{1, \ldots, n_a\}$ (there are n_a of them);
- Constraint #3 : makes sure that $IndB_i$ reflects whether data point B_i is included in the linear combination or not $\forall i \in \{1, \ldots, n_b\}$ (there are n_b of them);
- Constraint #4 : makes sure at least one point from the dataset is involved in the linear combination (there is a single one);
- Constraint #5 : makes sure that $TriA_i$ is equal to the weight value of data point A_i is that last is included in the linear combination, and 1 otherwise (there is a single one);
- Constraint #6 : makes sure that $TriB_i$ is equal to the weight value of data point B_i is that last is included in the linear combination, and 1 otherwise (there is a single one);

Dual model #1 explaination (2/2)

- Constraint #7 : making sure that the points that should not be included in the linear combination aren't part of it (there is a single one);
- Constraint #8 : making sure that the points that should not be included in the linear combination aren't part of it (there is a single one).

Dual MiniZinc model #1 (1/2)

n_1	Number of data points - f.c.
n_2	Number of data points - s.c.
d	Dimension of the data points
$pA \in \{0,1\}^{n_a}$	Datapoints not to include in the linear combination
$pB \in \{0,1\}^{n_b}$	Datapoints not to include in the linear combination
$A \in [0,1]^{n_a}$	Data points - f.c.
$B \in [0,1]^{n_b}$	Data points - s.c.
$ValA \in [0,1]^d$	Weights - f.c.
$ValB \in [0,1]^d$	Weights - s.c.
$\text{dom}(WhiA_i) = \{0,\dots,d\}$	$\forall i \in \{1,\dots,n_a\}$;
$\text{dom}(WhiB_i) = \{0,\dots,d\}$	$\forall i \in \{1,\dots,n_b\}$;
$\text{dom}(IndA_i) = \{0,1\}$	$\forall i \in \{1,\dots,n_a\}$;
$\text{dom}(IndB_i) = \{0,1\}$	$\forall i \in \{1,\dots,n_b\}$;
$\text{dom}(TriA_i) = [0,1]$	$\forall i \in \{1,\dots,n_a\}$;
$\text{dom}(TriB_i) = [0,1]$	$\forall i \in \{1,\dots,n_b\}$;
$\min_{i\in\{1,\dots,n_a\}} TriA_i + \min_{i\in\{1,\dots,n_b\}} TriB_i$	Objective function (maximize)

Dual MiniZinc model #1 (2/2)

$$\sum_{i\in\{1,\dots,n_a\}} WhiA_i * A_{i,j} + \sum_{i\in\{1,\dots,n_b\}} WhiB_i * B_{i,j} \qquad \forall j \in \{1,\dots,d\} \ (1)$$

$$(WhiA_j > 0 \land IndA_j = 1) \lor (WhiA_j = 0 \land IndA_j = 0) \qquad \forall i \in \{1,\dots,n_a\} \ (2)$$

$$(WhiB_j > 0 \land IndB_j = 1) \lor (WhiB_j = 0 \land IndB_j = 0) \qquad \forall i \in \{1,\dots,n_b\} \ (3)$$

$$\sum_{i\in\{1,\dots,n_a\}} IndA_i > 0 \qquad (4)$$

$$(IndA_i = 1 \land TriA_i = ValA_i) \lor (IndA_i = 0 \land TriA_i = V1) \qquad \forall i \in \{1,\dots,n_a\} \ (5)$$

$$(IndB_i = 1 \land TriB_i = ValB_i) \lor (IndB_i = 0 \land TriB_i = V1) \qquad \forall i \in \{1,\dots,n_b\} \ (6)$$

$$pA_i = 0 \lor IndA_i = 0 \qquad \forall i \in \{1,\dots,n_a\} \ (7)$$

$$pB_i = 0 \lor IndB_i = 0 \qquad \forall i \in \{1,\dots,n_b\} \ (8)$$

Now, some considerations:

- Notice that model 1 considers positive linear combination of points from the dataset with coefficients in $\{0,\dots,d\}^n, n \in n_a, n_b$ and not in $\mathbb{R}^n_+$ as in Lemma 1; see Corollary 1.
- In practice, we do not only feed to models #1 and #2 the resulting set of point minimizing the objective function of model #1, but also every intermediate sets of points. Doing so really makes the training faster.
- Model does not only find a solution satisying the contraint, is finds the non-linearly separable subset where the smallest weight is maximized. We do so because most of the time, only

unconsidering a single point suffices so that the remaining become linearly separable. So, no matter which point is removed, it will have an important impact on the obtainable accuracy.

Dual model #2 explaination

- n_1 : number of example of the first class;
- n_2 : number of example of the second class;
- d : dimension of the examples;
- c : number of subset of point that aren't linearly separable that has been spotted so far;
- pA : binary $n_a \times c$ matrix telling, on each row, which points from A are part of a non-linearly subset;
- pB : binary $n_b \times c$ matrix telling, on each row, which points from A are part of a non-linearly subset;
- A : $n_a \times d$ matrix containing every first class points;
- B : $n_b \times d$ matrix containing every first class points;
- *ValA* : vector of lenght n_a containing the weights of each datapoint of the first class;
- *ValB* : vector of lenght n_b containing the weights of each datapoint of the second class;
- Variables $IndA_i \forall i \in \{1, \ldots, n_a\}$: tells whether or not $WhiP_i$ is 0 or not;
- Variables $IndB_i \forall i \in \{1, \ldots, n_b\}$: tells whether or not $WhiM_i$ is 0 or not;
- Constraint #1 : making sure the solution yielded is not the same as one previously seen;

Dual MiniZinc model #2

n_1	Number of data points - f.c.
n_2	Number of data points - s.c.
d	Dimension of the data points
c	Nb. of non-lin. s. subsets seen
$pA \in \{0,1\}^{n_a \times c}$	Subsets not linearly separable
$pB \in \{0,1\}^{n_b \times c}$	Subsets not linearly separable
$A \in [0,1]^{n_a}$	Data points - f.c.
$B \in [0,1]^{n_b}$	Data points - s.c.
$ValA \in [0,1]^d$	Weights - f.c.
$ValB \in [0,1]^d$	Weights - s.c.
$\text{dom}(IndA_i) = \{0,1\}$	$\forall i \in \{1, \ldots, n_a\}$;
$\text{dom}(IndB_i) = \{0,1\}$	$\forall i \in \{1, \ldots, n_b\}$;
$\sum_{i \in 1,\ldots,n_a\}} IndA_i * ValA_i + \sum_{i \in 1,\ldots,n_a\}} IndA_i * ValA_i$	Objective function (maximize)
$\left(\bigvee_{i \in \{1,\ldots,n_a\}} (pA_{k,i} = 1 \wedge IndA_i = 0) \right)$	
$\vee \left(\bigvee_{i \in \{1,\ldots,n_b\}} (pB_{k,i} = 1 \wedge InBA_i = 0) \right)$	$\forall k \in \{1, \ldots, c\}$ (1)

4.3 Strong duality

Since both our approaches (primal and dual) lead to the single optimal solution, and that the primal constitutes a lower bound on the obtainable accuracy, and the dual an upper bound, because the accuracy diminishes throughout the training, then there is a **strong duality**:

Theorem 1. *Let* $\mathbf{A} \in \mathbb{R}^{d \times n_A} \times \{1\}^{n_A}$, $\mathbf{B} \in \mathbb{R}^{d \times n_B} \times \{1\}^{n_B}$ *be two sets of points from* $\mathbb{R}^d \times \{1\}$. *Let* $f(\mathbf{w}; \mathbf{C}, \mathbf{D}) = \sum_i \mathbb{1}\{\langle \mathbf{w}, \mathbf{c}_i \rangle < 0\} + \sum_j \mathbb{1}\{\langle \mathbf{w}, \mathbf{d}_j \rangle \geq 0\}$ *and* $LS(\mathbf{C}, \mathbf{D})$ *be true if* $\mathbf{C}$ *and* $\mathbf{D}$ *and linearly separable and false otherwise. The problem of finding the separating hyperplane achieving the lowest 0-1 loss lead to the two following optimization problems, forming a strong duality:*

Primal form

$$\underset{\mathbf{w}}{\text{maximize}} \quad f(\mathbf{w}; \mathbf{A}, \mathbf{B})$$

Dual form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & |\mathbf{C}| + |\mathbf{D}| \\ \text{subject to} \quad & LS(\mathbf{A} \setminus \mathbf{C}, \mathbf{B} \setminus \mathbf{D}) \end{aligned}$$

Proof. The primal form directly maximizes the 0-1 accuracy; the solution given by the primal form is optimal. The dual form yield a solution such that the number of points that aren't correctly classified is minimized. Therefore, it maximizes the 0-1 accuracy and yield an optimal solution. □

4.4 The CPNet algorithm

First of all, let's recap what the dual algorithm looks like.

Algorithm 1 The dual algorithm

Input : $\mathbf{A} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{n_a}, y_{n_a})\}$, $y \in \{+1\}$, $\mathbf{x} \in [0,1]^d \times \{1\}$, positive examples
  $\mathbf{B} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{n_b}, y_{n_b})\}$, $y \in \{-1\}$, $\mathbf{x} \in [0,1]^d \times \{1\}$, a null examples
  $M_{d,1}$ and $M_{d,2}$, the first and the second dual model
$\rho = True$
$t = 0$
$\mathbf{r} = \mathbf{0}_{n_a + n_b}$
While ρ :
  $t = t + 1$
  $\mathbf{T}_t = M_{d,1}(\mathbf{A}, \mathbf{B}; \mathbf{r})$
  If $|\mathbf{T}_t| = 0$:
    $\rho = False$
  Else:
    $\mathbf{r} = M_{d,1}(\mathbf{A}, \mathbf{B}; \mathbf{T}_1, \ldots, \mathbf{T}_t)$
Output : $\mathbf{r}$

Algorithm 1 shows the behaviour of the dual algorithm. $M_{d,1}$ seeks for sets of points that aren't linearly separable. Each time it is called, it can spot more than one, each of them being consigned in matrix $\mathbf{T}_t$. Those subsets are than given to $M_{d,2}$, trying to determine which points should be unconsidered so that the weighted sum of these points is minimized. Its selection is given to $M_{d,1}$, having to find subsets that do not involve any of these points, and so on.

Algorithm 2 presents the CPNet algorithm. We can separate each step of the algorithm into two main parts:

1. (line 12) The parametrization of a neuron is done. To do so, we make use of the fact that the primal approach can easily restrict the obtained parameterization to have at most two non-zero components (excluding a bias, so three total). Because of this constraint, this training phase can be done pretty quickly.
2. (lines 13 to 23) The (re)parameterization of the output layer is made. Notice on line 13 that a unit term is added, so that vector $\mathbf{c}$ contains an implicit bias. For that second part, the training can be quite long to do. So, we start both primal and dual approaches at the same time. There are two possibilities: either M_d converges, so that we can remove the points it has spotted from the dataset and compute a simple hard-margin SVM on the remaining linearly separable dataset; or we directly use the model given by M_p. This occurs when: the upper bound on the obtainable accuracy (given by M_d) is at ϵ from what M_p already

Algorithm 2 The CPNet algorithm

Input : $\mathbf{A} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{n_a}, y_{n_a})\}$, $y \in \{+1\}$, $\mathbf{x} \in [0,1]^d \times \{1\}$, positive examples
$\mathbf{B} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_{n_b}, y_{n_b})\}$, $y \in \{-1\}$, $\mathbf{x} \in [0,1]^d \times \{1\}$, a null examples
M_p and M_d, the primal and the dual model
δ a limit of time per neuron optimization
ϵ a tolerance between the upper and lower bounds
l, the maximum hidden layer width
$\mathbf{A}', \mathbf{B}', \mathbf{W}, \mathbf{c} = \mathbf{A}, \mathbf{B}, \mathbf{0}_{l \times d+1}, \mathbf{0}_{d+1}$
$B(\cdot) = \mathbf{c}\,[\mathbb{1}\{\mathbf{W} \times \cdot\}, 1]$
$t = 0$
While $t \leq l \wedge |\mathbf{A}| > 0 \wedge |\mathbf{B}| > 0$:
    $t = t + 1$
    $\mathbf{w}_t = M_p(\mathbf{A}', \mathbf{B}'; c = 2)$
    $\mathbf{A}^*, \mathbf{B}^* = [\mathbb{1}\{\mathbf{W}\mathbf{A}'\}, \mathbf{1}_{n_a}]\,, [\mathbb{1}\{\mathbf{W}\mathbf{B}'\}, \mathbf{1}_{n_b}]$
    Start $M_p(\mathbf{A}^*, \mathbf{B}^*; c = 3)$ and $M_d(\mathbf{A}^*, \mathbf{B}^*)$
    $a_p, a_d = M_p.perf(), M_d.perf()$
    $ts =$ current time
    While $a_d - a_p > \epsilon \wedge M_p$ and M_d haven't converged $\wedge$ current time - $ts < \delta$:
        Wait
    If M_d has converged:
        $\mathbf{c} =$ SVM$(\mathbf{A} \cup \mathbf{B} \setminus M_d.output())$
    Else
        $\mathbf{c} = M_p.output()$
    $\mathbf{A}', \mathbf{B}' = \mathbf{A}, \mathbf{B} \setminus$ the points correctly classified by B
Output : $B(\mathbf{x}) = \mathbf{c} \cdot [\mathbb{1}\{\mathbf{W}\mathbf{x}\}, 1]$

achieves, or too much time has passed trying to find the optimal solution (controlled by an hyperparameter δ), or M_p converges. Finally, the points that are currently classified correctly by the model are removed from the dataset for the next iteration, so we can focus on misclassified points.

5 Numerical experiments

For our numerical experiments, we used Coin-BC 2.10.8/1.17.7 solver. Our algorithms were written in Python with the help of the Minizinc package.

5.1 Relevance of both approaches

Figure 1 illustrates the relevance of both approaches when tackling a toy dataset and trying to find the optimal location for a separating hyperplane. The primal approach quickly finds a good solution, but it takes long before the solver makes sure that that solution is optimal. As for the dual approach, we clearly see that it continuously reduces the upper bound on the obtainable accuracy until it finds the optimal solution. This figure also illustrate that both approaches do converges to a same solution.

5.2 Test on a real-life datasets

Divorce dataset [40] Our first test is on a dataset concerning divorce. Couples were to fill a form with many questions, answering on a scale on 0 to 4. There were a total of 54 questions. Only two were kept in our model, achieving 99.45% accuracy on the dataset of 170 examples:

$$B(x_1, x_2) = \mathbb{1}\{2x_1 + 2x_2 > 5\},$$

where : x_1 = "We share the same views about being happy in our life with my spouse" and x_2 = "We're just starting a discussion before I know what's going on". The labels go as follow : $y = 0$: "Divorced" and $y = 1$: "Married". A truly simple model that is just big enough to achieves great performances. What we understand here is that there is a strong relationship between these two variables and not to be divorced. Moreover, if one variable has a really strong value, it doesn't matter

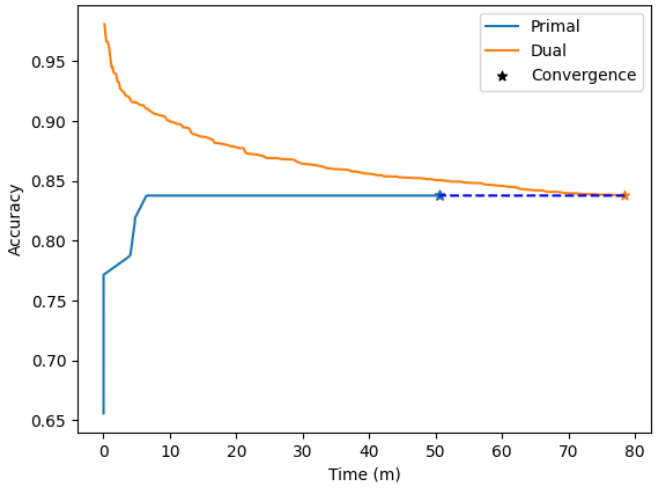

Figure 1: Training behaviour of both primal and dual approaches from placing a hyperplane in the feature space. Experiments on a dataset $\mathbf{A}, \mathbf{B} \in \{0,1\}^{50\times 11}$; 50 examples per classes, dimension 11 with random features.

whether or not the other one has a strong value. Since the dataset was small, only about five seconds were needed to build the predictor.

Cervical Cancer Behaviour Risk Data Set [41] This dataset contains 19 behaviour and feelings of people on a scale of 0 to 15, 21 of them having cervical cancer ($y = 1$) and 51 of them who hasn't ($y = 0$). We obtain the following model:

$$B(x_1, x_2) = \mathbb{1}\{\mathbb{1}\{x_1 + x_2 < 14\} + \mathbb{1}\{x_3 < 6\} > 0\}$$

Only three variables were kept in the model (x_1 = "Perception - Vulnerability", x_2 = "Empowerment - Abilities", x_3 = "Behaviour - Personal hygiene"), and still, it achieves 91.7% of accuracy. What we understand here is that variables x_1 and x_2 most be low-valued in order to predict that the person has cervical cancer. Not only that, but variable x_3 also need to be pretty low. In fact, if $x_1 + x_2$ is not small enough, or if x_3 is not small enough, than the prediction will be 0 - no cervical cancer. Both these relations thus play an important role in this model predictions. Since the dataset was small, only about fifteen seconds were needed to build the predictor.

6 Conclusion

We presented CPNet, an algorithm for building small, compact and interpretable binary activated neural networks. We use the word "building" because we start by considering a single hidden layer with a single neuron before iteratively making that layer grow. Such an approach leads to locally optimal components; for example, the output layer is always optimally optimized, and when a neuron is added, it is optimally placed in the feature space, but not network in its whole is not. Our approach was strongly dual, making use of two different perspectives of the problem of optimizing the 0-1 loss using a separating hyperplane. We also presented an interesting result giving a condition for a dataset to be linearly separable.

One of the most interesting improvement to the CPNet algorithm, in order to make sure the that model does not over-learn the training dataset, would be to include a criteria of relevance, as in decision tree growing (Gini index, for example), to decide whether it is worth it or not to add a new neuron to the hidden layer. The idea could be pushed even further: why not using such a criteria to decide whether it is relevant or not to have two or three non-zero weights on hidden neurons? Maybe, in some cases, only having a single non-zero weight on some neurons would yield a predictor as much as good while being more easily interpretable!

References

[1] Geoffrey Hinton, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.*, 29(6):82–97, 2012.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012.

[3] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[5] Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997, 2014.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778. IEEE Computer Society, 2016.

[7] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9. IEEE Computer Society, 2015.

[8] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *NIPS*, pages 4107–4115, 2016.

[9] Ji Lin, Chuang Gan, and Song Han. Defensive quantization: When efficiency meets robustness. *CoRR*, abs/1904.08444, 2019. URL `http://arxiv.org/abs/1904.08444`.

[10] Haotong Qin, Ruihao Gong, Xianglong Liu, Xiao Bai, Jingkuan Song, and Nicu Sebe. Binary neural networks: A survey. *Pattern Recognit.*, 105:107281, 2020.

[11] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *NIPS*, pages 3123–3131, 2015.

[12] Xiangming Meng, Roman Bachmann, and Mohammad Emtiyaz Khan. Training binary neural networks using the bayesian learning rule. In *ICML*, volume 119 of *Proceedings of Machine Learning Research*, pages 6852–6861. PMLR, 2020.

[13] Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *NIPS*, pages 963–971, 2014.

[14] Ziwei Wang, Jiwen Lu, Chenxin Tao, Jie Zhou, and Qi Tian. Learning channel-wise interactions for binary convolutional neural networks. In *CVPR*, pages 568–577. Computer Vision Foundation / IEEE, 2019.

[15] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *CoRR*, abs/1506.02626, 2015.

[16] Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *ICLR*. OpenReview.net, 2020.

[17] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *CoRR*, abs/1607.03250, 2016.

[18] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[19] Koen Goetschalckx, Bert Moons, Patrick Wambacq, and Marian Verhelst. Efficiently combining svd, pruning, clustering and retraining for enhanced neural network compression. In *EMDL@MobiSys*, pages 1–6. ACM, 2018.

[20] Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 2285–2294. JMLR.org, 2015.

[21] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. In *BMVC*. BMVA Press, 2014.

[22] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*. OpenReview.net, 2019.

[23] Eugene Belilovsky, Michael Eickenberg, and Edouard Oyallon. Greedy layerwise learning can scale to imagenet. In *ICML*, volume 97 of *Proceedings of Machine Learning Research*, pages 583–593. PMLR, 2019.

[24] Mandar Kulkarni and Shirish Subhash Karande. Layer-wise training of deep networks using kernel similarity. *CoRR*, abs/1703.07115, 2017.

[25] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *NIPS*, pages 153–160. MIT Press, 2006.

[26] Anton Dereventsov, Armenak Petrosyan, and Clayton G. Webster. Greedy shallow networks: A new approach for constructing and training neural networks. *CoRR*, abs/1905.10409, 2019.

[27] Elias B. Khalil, Amrita Gupta, and Bistra Dilkina. Combinatorial attacks on binarized neural networks. *CoRR*, abs/1810.03538, 2018.

[28] Chih-Hong Cheng, Georg Nührenberg, Chung-Hao Huang, and Harald Ruess. Verification of binarized neural networks via inter-neuron factoring - (short paper). In *VSTTE*, volume 11294 of *Lecture Notes in Computer Science*, pages 279–290. Springer, 2018.

[29] Rudy Bunel, Ilker Turkaslan, Philip H. S. Torr, Pushmeet Kohli, and Pawan Kumar Mudigonda. A unified view of piecewise linear neural network verification. In *NeurIPS*, pages 4795–4804, 2018.

[30] Nina Narodytska, Shiva Prasad Kasiviswanathan, Leonid Ryzhyk, Mooly Sagiv, and Toby Walsh. Verifying properties of binarized deep neural networks. In *AAAI*, pages 6615–6624. AAAI Press, 2018.

[31] Rodrigo Toro Icarte, León Illanes, Margarita P. Castro, André A. Ciré, Sheila A. McIlraith, and J. Christopher Beck. Training binarized neural networks using MIP and CP. In *CP*, volume 11802 of *Lecture Notes in Computer Science*, pages 401–417. Springer, 2019.

[32] Han Zhou, Aida Ashrafi, and Matthew B. Blaschko. Combinatorial optimization for low bit-width neural networks. In *ICPR*, pages 2246–2252. IEEE, 2022.

[33] Wenrui Hao, Xianlin Jin, Jonathan W. Siegel, and Jinchao Xu. An efficient greedy training algorithm for neural networks and applications in pdes. *CoRR*, abs/2107.04466, 2021.

[34] Leonardo Lucio Custode, Ciro Lucio Tecce, Illya Bakurov, Mauro Castelli, Antonio Della Cioppa, and Leonardo Vanneschi. A greedy iterative layered framework for training feed forward neural networks. In *EvoApplications*, volume 12104 of *Lecture Notes in Computer Science*, pages 513–529. Springer, 2020.

[35] Benjamin Leblanc and Pascal Germain. A greedy algorithm for building compact binary activated neural networks. *CoRR*, abs/2209.03450, 2022.

[36] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead, 2019.

[37] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *CoRR*, abs/2103.11251, 2021.

[38] Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should I trust you?": Explaining the predictions of any classifier. In *KDD*, pages 1135–1144. ACM, 2016.

[39] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *NIPS*, pages 4765–4774, 2017.

[40] Mustafa Kemal Yöntem, Kemal Adem, Tahsin İlhan, and Serhat Kılıçarslan. Divorce prediction using correlation based feature selection and artificial neural networks. *Nevşehir Hacı Bektaş Veli Üniversitesi SBE Dergisi*, 9(1):259 – 273, 2019.

[41] . Sobar, Rizanda Machmud, and Adi Wijaya. Behavior determinant based cervical cancer early detection with machine learning algorithm. *Advanced Science Letters*, 22(10):3120–3123, 2016. ISSN 1936-6612. doi: doi:10.1166/asl.2016.7980. URL `https://www.ingentaconnect.com/content/asp/asl/2016/00000022/00000010/art00111`.

Appendix

A Mathematical results

A.1 Proof of Lemma 1

Lemma 1 (Linear separability criteria)**.**
Let $\mathbf{A} \in \mathbb{R}^{d \times n_A} \times \{1\}^{n_A}$, $\mathbf{B} \in \mathbb{R}^{d \times n_B} \times \{1\}^{n_B}$ *be two sets of points from* $\mathbb{R}^d \times \{1\}$. Let $LS(\mathbf{A}, \mathbf{B})$ *be true if* $\mathbf{A}$ *and* $\mathbf{B}$ *are linearly separable, and false otherwise. Then:*

$$LS(A, B) \Leftrightarrow (\nexists \mathbf{c} \in \mathbb{R}^{n_A}, \mathbf{d} \in \mathbb{R}^{n_B} \mid \mathbf{Ac} = \mathbf{Bd} \neq \mathbf{0}) .$$

Proof. Let us first remind of Farka's lemma:

Lemma 2 (Farka's lemma)**.** *Let* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *and* $\mathbf{b} \in \mathbb{R}^m$*. Then exactly one of the following two assertions is true:*

1. *There exists an* $\mathbf{x} \in \mathbb{R}^n$ *such that* $\mathbf{Ax} = \mathbf{b}$ *and* $\mathbf{x} \geq 0$*;*
2. *There exists a* $\mathbf{y} \in \mathbb{R}^m$ *such that* $\mathbf{A}^T\mathbf{y} \geq 0$ *and* $\mathbf{b}^T\mathbf{y} < 0$*.*

Let us also remind of a version of the hyperplane separation theorem involving both a closed set and an open set:

Theorem 2 (Hyperplane separation theorem)**.** *Let* A *and* B *be two disjoint nonempty convex sets. If* A *is open, then there exist a non-zero vector* $\mathbf{v}$ *such that*

$$(\forall \mathbf{a} \in A \mid \langle \mathbf{a}, \mathbf{v} \rangle \geq 0) ,$$
$$(\forall \mathbf{b} \in B \mid \langle \mathbf{b}, \mathbf{v} \rangle < 0) .$$

Here, the notation $\mathbf{x} \geq 0$ means that all components of the vector $\mathbf{x}$ are non-negative, and $\mathbf{x} > 0$ that they're all positive. Let $B = \{\mathbf{b} \mid \left(\exists \mathbf{c} \in \mathbb{R}_+^{n_B} \setminus \{\mathbf{0}_{n_B}\} \mid \mathbf{b} = \mathbf{Bc}\right)\}$. We have that

$$\begin{aligned}
LS(\mathbf{A}, \mathbf{B}) &\Leftrightarrow \left(\exists \mathbf{y} \in \mathbb{R}^d \mid \mathbf{A}^T\mathbf{y} \geq 0 \wedge \mathbf{B}^T\mathbf{y} < 0\right) && \langle \text{By definition} \rangle \\
&\Leftrightarrow \left(\exists \mathbf{y} \in \mathbb{R}^d \mid \mathbf{A}^T\mathbf{y} \geq 0 \wedge \left(\forall \mathbf{b} \in B \mid \mathbf{b}^T\mathbf{y} < 0\right)\right) && \langle \text{Positive linear combination of} \\
& && \text{negative terms yield a negative term} \rangle \\
&\Leftrightarrow \left(\exists \mathbf{y} \in \mathbb{R}^d \mid \left(\forall \mathbf{b} \in B \mid \mathbf{A}^T\mathbf{y} \geq 0 \wedge \mathbf{b}^T\mathbf{y} < 0\right)\right) && \langle \text{Conjonction's distributive prop.} \rangle \\
&\Leftrightarrow \left(\forall \mathbf{b} \in B \mid \left(\exists \mathbf{y} \in \mathbb{R}^d \mid \mathbf{A}^T\mathbf{y} \geq 0 \wedge \mathbf{b}^T\mathbf{y} < 0\right)\right) && \langle (1) \rangle \\
&\Leftrightarrow \left(\forall \mathbf{b} \in B \mid \left(\nexists \mathbf{x} \in \mathbb{R}_+^{n_A} \mid \mathbf{Ax} = \mathbf{b}\right)\right) && \langle \text{Farka's lemma} \rangle \\
&\Leftrightarrow (\nexists \mathbf{c} \in \mathbb{R}^{n_A}, \mathbf{d} \in \mathbb{R}^{n_B} \mid \mathbf{Ac} = \mathbf{Bd} \neq \mathbf{0}) && \langle \text{Arithmetic, definition of } B \rangle
\end{aligned}$$

We'll have to show that (1) is true; while the right implication is trivial:

$\left(\exists \mathbf{y} \in \mathbb{R}^d \mid \left(\forall \mathbf{b} \in B \mid \mathbf{A}^T\mathbf{y} \geq 0 \wedge \mathbf{b}^T\mathbf{y} < 0\right)\right) \Rightarrow \left(\forall \mathbf{b} \in B \mid \left(\exists \mathbf{y} \in \mathbb{R}^d \mid \mathbf{A}^T\mathbf{y} \geq 0 \wedge \mathbf{b}^T\mathbf{y} < 0\right)\right)$,

the left one is more subtle:

$\left(\forall \mathbf{b} \in B \mid \left(\exists \mathbf{y} \in \mathbb{R}^d \mid \mathbf{A}^T\mathbf{y} \geq 0 \wedge \mathbf{b}^T\mathbf{y} < 0\right)\right) \Rightarrow \left(\exists \mathbf{y} \in \mathbb{R}^d \mid \left(\forall \mathbf{b} \in B \mid \mathbf{A}^T\mathbf{y} \geq 0 \wedge \mathbf{b}^T\mathbf{y} < 0\right)\right)$.

Let A be the convex hull of $\mathbf{A}$. Notice that B can be seen as a convex cone with its vertex being open. The left term here implicitly tells us that these two convex sets are disjoint. Indeed, any point in B is linearly separable from A, so there are no point of B in A. Therefore, the Hyperplane separation theorem tells us that a single hyperplane (parameterized by $\mathbf{y}$) can separate B from A, or in other words, any point from B from $\mathbf{A}$. □

Corollary 1 (Linear separability criteria - Binary features)**.**
Let $\mathbf{A} \in \{0,1\}^{d \times n_A} \times \{1\}^{n_A}$, $\mathbf{B} \in \{0,1\}^{d \times n_B} \times \{1\}^{n_B}$ *be two sets of points from* $\mathbb{R}^d \times \{1\}$. *Let* $LS(\mathbf{A}, \mathbf{B})$ *be true if* $\mathbf{A}$ *and* $\mathbf{B}$ *are linearly separable, and false otherwise. Then:*

$$LS(A, B) \Leftrightarrow (\nexists \mathbf{c} \in \{0, \ldots, d-1\}^{n_A}, \mathbf{d} \in \{0, \ldots, d-1\}^{n_B} \mid \mathbf{Ac} = \mathbf{Bd} \neq \mathbf{0}) \,.$$

B Extended details on the approach

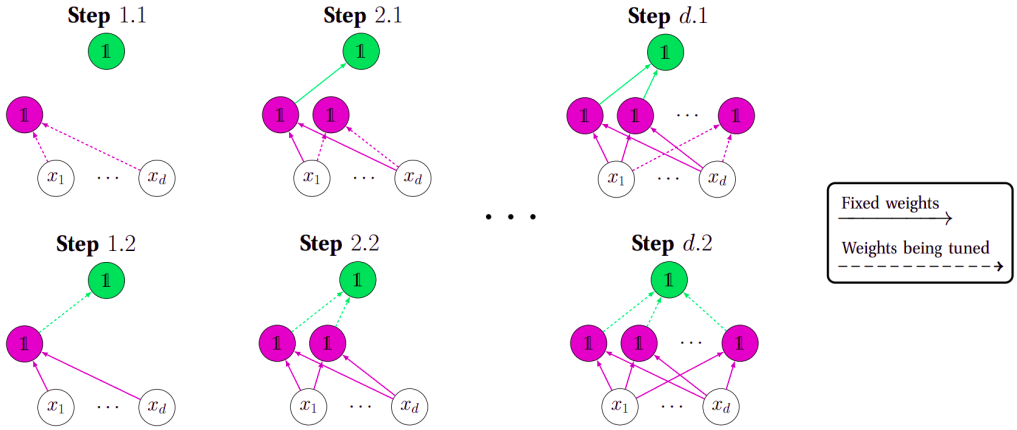

Figure 2: Visual representation of the steps for building of a neural network following the CPNet algorithm.

Figure 2 represents the steps for building a BANN following the CPNet algorithm. The purple weights and neurons represent the hidden layer and its parameterization; the green weights and neuron correspond to the output layer. Since we tackle binary classification tasks, the output layer only has a single neuron and a threshold (indicator) activation function. Indeed, the resulting BANN has a single hidden layer of size d. Each time a neuron is added, its parameterization is computed so that it is locally optimal. Then, the output layer weights and bias is updated so that the whole output layer is optimal at the end of each step.