Training Binarized Neural Networks using MIP and CP*

Rodrigo Toro Icarte[1,2], León Illanes[1], Margarita P. Castro[3], Andre A. Cire[4], Sheila A. McIlraith[1,2], and J. Christopher Beck[3]

[1] Department of Computer Science, University of Toronto, Toronto, Canada
{rntoro,lillanes,sheila}@cs.toronto.edu
[2] Vector Institute, Toronto, Canada
[3] Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada {mpcastro,jcb}@mie.utoronto.ca
[4] Department of Management, University of Toronto Scarborough, Toronto, Canada
acire@utsc.utoronto.ca

Abstract. Binarized Neural Networks (BNNs) are an important class of neural network characterized by weights and activations restricted to the set $\{-1, +1\}$. BNNs provide simple compact descriptions and as such have a wide range of applications in low-power devices. In this paper, we investigate a model-based approach to training BNNs using constraint programming (CP), mixed-integer programming (MIP), and CP/MIP hybrids. We formulate the training problem as finding a set of weights that correctly classify the training set instances while optimizing objective functions that have been proposed in the literature as proxies for generalizability. Our experimental results on the MNIST digit recognition dataset suggest that—when training data is limited—the BNNs found by our hybrid approach generalize better than those obtained from a state-of-the-art gradient descent method. More broadly, this work enables the analysis of neural network performance based on the availability of optimal solutions and optimality bounds.

Keywords: Binarized Neural Networks · Machine Learning · Constraint Programming · Mixed Integer Programming · Discrete Optimization.

1 Introduction

Deep learning is responsible for recent breakthroughs in image recognition, speech recognition, language translation, and artificial intelligence [18, 7]. Roughly speaking, deep learning aims to find a set of weights for a neural network (NN) that maps training inputs to target outputs (e.g., English sentences to their Spanish translations), a process known as training. The most notable feature of deep learning is that NNs *generalize* when trained over large datasets, i.e., they can map unseen inputs to their target outputs with high accuracy.

* This is the authors' copy of a paper that is to appear in the proceedings of the 25th International Conference on Principles and Practice of Constraint Programming.

Hubara et al. [9] recently showed that Binarized Neural Networks (BNNs)—NNs with weights and activations in $\{-1, +1\}$—have comparable test performance to standard NNs in two well-known image recognition datasets. This is a remarkable result because BNNs can be implemented using Boolean operations with low memory and energy consumption, enabling, for example, the application of deep learning in mobile devices. While training BNNs is a discrete optimization problem, it has not been addressed by model-based techniques such as mixed-integer programming (MIP) or constraint programming (CP). Instead, BNNs are trained using gradient descent (GD) methods over continuous weights which are binarized during the forward pass of the algorithm [26, 20, 33, 17].

Model-based approaches have stronger convergence guarantees than GD and, as such, can potentially find better solutions given enough time and resources. There are two reasons, however, why a model-based approach—in particular MIP—may be disadvantageous, as stated by Gambella et al. [6]. First, it may not scale to large datasets since the size of the model depends on the size of the training set. Second, solutions with provably-optimal training error are likely to *overfit* the data, that is, they will classify the training examples effectively but will not generalize.

The main contribution of this paper is a collection of model-based training methods that explicitly address these issues. The key insights are (i) improving scalability by taking advantage of CP's ability to find BNNs that fit the training data and (ii) avoiding overfitting by optimizing well-known proxies for generalizability. Specifically, we propose MIP, CP, and CP/MIP hybrid approaches to train BNNs while optimizing objective functions based on two machine learning principles for generalization: *simplicity* and *robustness*.

We experimented over subsets of the widely-used MNIST dataset [19]. Our experiments focused on limited training data, a setting known as *few-shot learning* [32]. This setting is important in Machine Learning because collecting labeled data is expensive—or even impossible—in many important real-world applications, including healthcare [21, 3]. Our results show that our hybrid methods scale significantly better than MIP (i.e., they solve problems with larger networks and more training data) and produced BNNs that generalize better than those trained with GD. In fact, our BNNs correctly classified up to 3 times more unseen examples than BNNs learned by GD on a few-shot learning regime. However, model-based approaches are still far from scaling at the level of GD and, hence, GD should be preferred when a large amount of data is available. Finally, since model-based approaches find provably-optimal solutions—GD does not—they allow for principled empirical comparisons between generalization proxies. Our results suggest that optimizing for robustness leads to better test performance than simplicity.

2 Problem Definition

BNNs are NNs with weights and activations restricted to the values -1 and $+1$. A BNN architecture is defined by the number of layers L and the set of

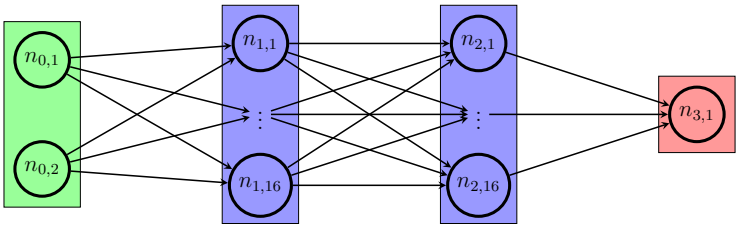


Fig. 1: A BNN with 2 inputs, two hidden layers with 16 neurons each, and 1 output neuron. We use the notation $n_{\ell j}$ to represent neuron j from layer ℓ.

neurons $\mathcal{N} = \langle N_0, \ldots, N_L \rangle$, where N_ℓ corresponds to the set of neurons in layer $\ell \in \{0, .., L\}$. For instance, Figure 1 shows a BNN with two input neurons ($n_{0,1}$ and $n_{0,2}$), two hidden layers with 16 neurons each ($n_{1,1}$ to $n_{1,16}$ and $n_{2,1}$ to $n_{2,16}$), and one output neuron ($n_{3,1}$), i.e., its architecture is $\mathcal{N} = \langle N_0, N_1, N_2, N_3 \rangle$ with $|N_0| = 2$, $|N_1| = |N_2| = 16$, and $|N_3| = 1$. Every neuron $j \in N_\ell$ ($\ell \geq 1$) is connected to every neuron $i \in N_{\ell-1}$ by a weight $w_{i\ell j} \in \{-1, 0, 1\}$. Note that setting $w_{i\ell j} = 0$ is equivalent to removing the corresponding connection from the BNN. Given a value $\mathbf{x}$ for the input neurons, the *preactivation* $a_{\ell j}(\mathbf{x})$ of neuron $j \in N_\ell$ and its *activation* $n_{\ell j}(\mathbf{x})$ are, respectively,

$$a_{\ell j}(\mathbf{x}) = \sum_{i \in N_{\ell-1}} w_{i\ell j} \cdot n_{(\ell-1)i}(\mathbf{x}) \qquad \text{and} \qquad n_{\ell j}(\mathbf{x}) = \begin{cases} \mathbf{x}_j & \text{if } \ell = 0 \\ +1 & \text{if } \ell > 0,\ a_{\ell j}(\mathbf{x}) \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

The activations of all the neurons in a BNN are -1 or $+1$ except for the input neurons, which may take any real value. A weight assignment $\mathbf{W}$ to the network defines a function $f_{\mathbf{W}} : \mathbb{R}^{|N_0|} \to \{-1, 1\}^{|N_L|}$ that maps input vectors $\mathbf{x} \in \mathbb{R}^{|N_0|}$ to output vectors $\mathbf{y} \in \{-1, 1\}^{|N_L|}$, where $\mathbf{y}$ represents the neuron activations in the last layer. *Training* a BNN consists of finding a weight assignment that fits a training set $\mathcal{T} = \langle (\mathbf{x}^1, \mathbf{y}^1), \ldots, (\mathbf{x}^\tau, \mathbf{y}^\tau) \rangle$, i.e., finding $\mathbf{W}$ such that $f_{\mathbf{W}}(\mathbf{x}^k) = \mathbf{y}^k$ for all pairs $(\mathbf{x}^k, \mathbf{y}^k) \in \mathcal{T}$. The task of learning functions from input-output examples is known as *supervised learning*.

The goal of supervised learning is *generalization* [4]. A trained BNN is useful only if it can map unseen examples to their correct outputs (i.e., good test performance). Hence, a central problem is how to distinguish BNNs that generalize from those that overfit the training data. There are two main principles to avoid overfitting in machine learning (ML): simplicity and robustness.

The simplicity principle, also known as Occam's razor, suggests that we should prefer the simplest BNNs that fit the training set. For NNs, a natural measure of simplicity is the number of connections [23]. Our first optimization problem, therefore, looks for a BNN that fits the training data and minimizes the number of nonzero weights:

$$\min_{\mathbf{W}} \left\{ \sum_{w \in \mathbf{W}} |w| \colon f_{\mathbf{W}}(\mathbf{x}) = \mathbf{y},\ \forall (\mathbf{x}, \mathbf{y}) \in \mathcal{T},\ w \in \{-1, 0, 1\},\ \forall w \in \mathbf{W} \right\}. \quad \text{(min-weight)}$$

While the effectiveness of this principle has been challenged [4], it is the basis for most forms of regularizers used in modern deep learning [27].

In contrast, the robustness principle looks for BNNs that classify the training set correctly despite small perturbations to their weights. It is believed that deep NNs avoid overfitting because GD implicitly drives the exploration toward robust solutions [12, 13, 25]. One way of finding robust BNNs is by maximizing the *margins* of their neurons. Given a training set $\mathcal{T}$, the margin of neuron $n_{\ell j}$ is equal to the minimum absolute value of its preactivation $a_{\ell j}(\mathbf{x})$ for any $(\mathbf{x}, \mathbf{y}) \in \mathcal{T}$. Intuitively, neurons with larger margins require bigger changes on their inputs and weights to change their activation values. Recent work shows that margins are good predictors for the generalization of deep convolutional NNs [11]. Our second optimization problem searches for BNNs that fit the training data and have the maximum sum of neuron margins:

$$\begin{aligned} \max_{\mathbf{W}} \quad & \sum_{\ell \in \{1..L\}} \sum_{j \in N_\ell} \min\{|a_{\ell j}(\mathbf{x})| : (\mathbf{x}, \mathbf{y}) \in \mathcal{T}\} && \text{(max-margin)} \\ \text{s.t.} \quad & f_{\mathbf{W}}(\mathbf{x}) = \mathbf{y} && \forall (\mathbf{x}, \mathbf{y}) \in \mathcal{T} \\ & w \in \{-1, 0, 1\} && \forall w \in \mathbf{W} \end{aligned}$$

We focus on these two criteria because they are well-supported by previous work. However, there are likely to be other objective functions worth studying and our models may be extended to do so. Additionally, our models assume that the training set has no incorrectly labeled training examples. Extensions to handle noise can be done by including slack variables as proposed in the Support Vector Machine literature [29].

3 Related Work

Unfortunately, BNNs cannot be trained using standard backpropagation because their weights are discrete. Hubara et al. [9] proposed using two sets of weights: $\mathbf{W}$ and $\mathbf{W}_b$, with $\mathbf{W}$ taking continuous values. When computing the activations, the weights $\mathbf{W}$ and activations a are projected to -1 or $+1$ using $\mathbf{W}_b = \text{sign}(\mathbf{W})$ and $a_b = \text{sign}(a)$. Then, the gradients are computed as usual except for the activation function. To backpropagate over $\text{sign}(a)$ they assume that its gradient is equal to 1 if $|a| \leq 1$ and is 0 otherwise. These gradients update $\mathbf{W}$ and then the process repeats. While most work on training BNNs follows this approach [20, 26, 31, 33], there are a few gradient-based alternatives such as *Apprentice* [22] and *Self-Binarizing Networks* [17].

Other work has explored the use of model-based approaches, in particular MIP, in tasks related to NNs [6]. For example, MIP models have been proposed for NN verification [1] and for finding adversarial examples for NNs [5, 30] and BNNs [15]. Given a pre-trained network and a target input, the problem of finding an adversarial example consists of discovering the smallest perturbation of the target input such that the output of the network changes. In particular, Khalil et al. proposed a MIP model that, similarly to our work, uses big-M constraints to model the neuron activations [15]. They recognize those big-M constraints as the main bottleneck in scaling their approach and propose a heuristic

method that finds adversarial examples by fixing different subsets of the activations over time. One of our hybrid CP/MIP models, `HA`, exploits a similar idea but for training BNNs. SAT models have also been used in the context of verifying properties over BNNs [24, 2].

With regards to training BNNs, Khalil and Dilkina discussed the viability of using MIP models in an extended abstract at CPAIOR 2018 [14]. They report no specific results, but suggest that their MIP approach fails to scale. To the best of our knowledge, there is no other work on training BNNs using MIP or CP nor on any applications of CP to BNNs.

4 Monolithic Models For Training BNNs

We now introduce CP and MIP models to train BNNs. The models receive the training set $\mathcal{T} = \langle(\mathbf{x}^1, \mathbf{y}^1), \ldots, (\mathbf{x}^\tau, \mathbf{y}^\tau)\rangle$ and the network's architecture $\mathcal{N} = \langle N_0, \ldots, N_L\rangle$ as input. Our models use $T = \{1, \ldots, \tau\}$ as the set of training indices and $\mathcal{L} = \{1, \ldots, L\}$ as the set of layers.

4.1 Constraint Programming Models

Our CP models use the formalism and global constraints available in IBM ILOG CP Optimizer [10]. Let $w_{i\ell j} \in \{-1, 0, 1\}$ be a decision variable indicating the weight of the connection going from neuron $i \in N_{\ell-1}$ to $j \in N_\ell$. Let $n^k_{\ell j}$ be a CP expression representing the activation of neuron j in layer ℓ when the training instance $\mathbf{x}^k$ is fed to the BNN. Our model uses the vector notation $\mathbf{w}_{\ell j} = [w_{1\ell j}, ..., w_{|N_{\ell-1}|\ell j}]^\top$ and $\mathbf{n}^k_\ell = [n^k_{\ell 1}, ..., n^k_{\ell|N_\ell|}]^\top$. The constraints are:

$$n^k_{0j} = x^k_j \qquad \forall j \in N_0, k \in T \qquad (1)$$

$$n^k_{\ell j} = 2\left(\texttt{scal\_prod}(\mathbf{w}_{\ell j}, \mathbf{n}^k_{\ell-1}) \geq 0\right) - 1 \qquad \forall \ell \in \mathcal{L} \setminus \{L\}, j \in N_\ell, k \in T \qquad (2)$$

$$n^k_{Lj} = y^k_j \qquad \forall j \in N_L, k \in T \qquad (3)$$

$$w_{i\ell j} \in \{-1, 0, 1\} \qquad \forall \ell \in \mathcal{L}, i \in N_{\ell-1}, j \in N_\ell \qquad (4)$$

The first three constraints recursively define the neuron activations. Constraint (1) instantiates N_0 to be the same as the input vector for each training example. Constraint (2) defines the activations for the remaining layers, which depend on the variables $w_{i\ell j}$. This constraint uses a reified scalar product constraint, $\texttt{scal\_prod}(\mathbf{v}_1, \mathbf{v}_2) = \mathbf{v}_1^\top \cdot \mathbf{v}_2$, to compute the neuron activation. Constraint (3) matches the last neuron layer values to the output vector of each training example. Constraint (4) defines the variable domains.

Our CP models have identical sets of constraints but different objectives. Model $\texttt{CP}_w$ minimizes the total number of weights using the expression $\texttt{abs}(a) = |a|$ for absolute value, i.e.,

$$\min \sum_{\ell \in \mathcal{L}} \sum_{i \in N_{\ell-1}} \sum_{j \in N_\ell} \texttt{abs}(w_{i\ell j}), \quad \text{s.t. (1)–(4)}, \qquad (\texttt{CP}_w)$$

while model $\texttt{CP}_m$ maximizes the sum of neuron margins, i.e.,

$$\max \sum_{\ell \in \mathcal{L}} \sum_{j \in N_\ell} \min \Big(\{ \texttt{abs}(\texttt{scal\_prod}(\mathbf{w}_{\ell j}, \mathbf{n}^k_{\ell-1})) |\ k \in T \} \Big), \ \text{s.t. (1)–(4).} \qquad (\texttt{CP}_m)$$

Each CP model has $O(W)$ decision variables and $O(|N_L| \cdot \tau)$ constraints, where W is the number of weights, $|N_L|$ is the number of output neurons, and τ is the size of the training set.

4.2 Mixed Integer Programming Models

The MIP and CP models share the same main decision variables. Variable $w_{i\ell j} \in \{-1, 0, 1\}$ indicates the weight of the connection from neuron $i \in N_{\ell-1}$ to neuron $j \in N_\ell$. Variable $u^k_{\ell j} \in \{0, 1\}$ models the activation of neuron $j \in N_\ell$ when the training instance $\mathbf{x}^k$ is fed to the BNN. Note that the actual neuron activation is $n^k_{\ell j} = 2u^k_{\ell j} - 1$ in this case. In addition, we use an auxiliary variable to model the non-linearities inside the BNN. Variable $c^k_{i\ell j} \in \mathbb{R}$ represents the multiplication of neuron activation $i \in N_{\ell-1}$ for a given $k \in T$ and weight $w_{i\ell j}$, i.e., $c^k_{i\ell j} = (2u^k_{(\ell-1)i} - 1) \cdot w_{i\ell j}$. Lastly, we use sets $\mathcal{L}_2 = \{2, \ldots, L\}$ and $\mathcal{L}^{L-1} = \{1, \ldots, L-1\}$, and a small constant $\epsilon > 0$ to model strict inequalities.

Our minimum-weight $\texttt{MIP}_w$ model introduces a binary variable $v_{i\ell j} \in \{0, 1\}$ to represent the absolute value of each weight $w_{i\ell j}$. Constraints (5) and (6) force the BNN output to be equal to target value y^k_j in the training set. Constraints (7) and (8) are implication constraints (which can be reformulated as big-M constraints) that define the activations. Constraint (9) sets the value of c^k_{i1j} for the input layer, while constraints (10) to (13) ensure that $c^k_{i\ell j} = (2u^k_{(\ell-1)i} - 1) \cdot w_{i\ell j}$. Constraint (14) defines the absolute values of each weight. Lastly, constraints (15) to (18) specify the domains of the variables.

$$\min \sum_{\ell \in \mathcal{L}} \sum_{i \in N_{\ell-1}} \sum_{j \in N_\ell} v_{i\ell j} \qquad (\texttt{MIP}_w)$$

$$\text{s.t.} \sum_{i \in N_{L-1}} c^k_{iLj} \geq 0 \qquad \forall j \in N_L, k \in T : y^k_j = 1 \qquad (5)$$

$$\sum_{i \in N_{L-1}} c^k_{iLj} \leq -\epsilon \qquad \forall j \in N_L, k \in T : y^t_j = -1 \qquad (6)$$

$$(u^k_{\ell j} = 1) \implies \left(\sum_{i \in N_{\ell-1}} c^k_{i\ell j} \geq 0 \right) \qquad \forall \ell \in \mathcal{L}^{L-1}, j \in N_\ell, k \in T \qquad (7)$$

$$(u^k_{\ell j} = 0) \implies \left(\sum_{i \in N_{\ell-1}} c^k_{i\ell j} \leq -\epsilon \right) \qquad \forall \ell \in \mathcal{L}^{L-1}, j \in N_\ell, k \in T \qquad (8)$$

$$c^k_{i1j} = x^k_i \cdot w_{i1j} \qquad \forall i \in N_0, j \in N_1, k \in T \qquad (9)$$

$$c^k_{i\ell j} - w_{i\ell j} + 2u^k_{(\ell-1)i} \leq 2 \qquad \forall \ell \in \mathcal{L}_2, i \in N_{\ell-1}, j \in N_\ell, k \in T \qquad (10)$$

$$c^k_{i\ell j} + w_{i\ell j} - 2u^k_{(\ell-1)i} \leq 0 \qquad \forall \ell \in \mathcal{L}_2, i \in N_{\ell-1}, j \in N_\ell, k \in T \qquad (11)$$

$$c^k_{i\ell j} - w_{i\ell j} - 2u^k_{(\ell-1)i} \geq -2 \qquad \forall \ell \in \mathcal{L}_2, i \in N_{\ell-1}, j \in N_\ell, k \in T \qquad (12)$$

$$c^k_{i\ell j} + w_{i\ell j} + 2u^k_{(\ell-1)i} \geq 0 \qquad \forall \ell \in \mathcal{L}_2, i \in N_{\ell-1}, j \in N_\ell, k \in T \quad (13)$$

$$-v_{i\ell j} \leq w_{i\ell j} \leq v_{i\ell j} \qquad \forall \ell \in \mathcal{L}, i \in N_{\ell-1}, j \in N_\ell \quad (14)$$

$$w_{i\ell j} \in \{-1, 0, 1\} \qquad \forall \ell \in \mathcal{L}, i \in N_{\ell-1}, j \in N_\ell \quad (15)$$

$$u^k_{\ell j} \in \{0, 1\} \qquad \forall \ell \in \mathcal{L}^{L-1}, j \in N_\ell, k \in T \quad (16)$$

$$c^k_{i\ell j} \in \mathbb{R} \qquad \forall \ell \in \mathcal{L}, i \in N_{\ell-1}, j \in N_\ell, k \in T \quad (17)$$

$$v_{i\ell j} \in \{0, 1\} \qquad \forall \ell \in \mathcal{L}, i \in N_{\ell-1}, j \in N_\ell \quad (18)$$

The maximum-margin $\texttt{MIP}_m$ model introduces a variable $m_{\ell j} \in \mathbb{R}^+$ to represent the margin of each neuron $j \in N_\ell$. The set of constraints is similar to the previous model with the exception that it includes neuron margin variables in the neuron activation constraints (19)–(22).

$$\max \sum_{\ell \in \mathcal{L}} \sum_{j \in N_\ell} m_{\ell j} \qquad (\texttt{MIP}_m)$$

$$\text{s.t. } (9)\text{–}(13), (15)\text{–}(17)$$

$$\sum_{i \in N_{L-1}} c^k_{iLj} \geq m_{Lj} \qquad \forall j \in N_L, k \in T : y^k_j = 1 \quad (19)$$

$$\sum_{i \in N_{L-1}} c^k_{iLj} \leq -\epsilon - m_{Lj} \qquad \forall j \in N_L, k \in T : y^t_j = -1 \quad (20)$$

$$(u^k_{\ell j} = 1) \implies \left(\sum_{i \in N_{\ell-1}} c^k_{i\ell j} \geq m_{\ell j} \right) \qquad \forall \ell \in \mathcal{L}^{L-1}, j \in N_\ell, k \in T \quad (21)$$

$$(u^k_{\ell j} = 0) \implies \left(\sum_{i \in N_{\ell-1}} c^k_{i\ell j} \leq -\epsilon - m_{\ell j} \right) \qquad \forall \ell \in \mathcal{L}^{L-1}, j \in N_\ell, k \in T \quad (22)$$

$$m_{lj} \geq 0 \qquad \forall \ell \in \mathcal{L}, j \in N_\ell \quad (23)$$

Note that each MIP model has $O(W + N \cdot \tau)$ integer decision variables and $O((W + N)\tau)$ constraints, where W is the number of weights, N is the total number of neurons, and τ is the size of the training set.

5 CP/MIP Hybrid Approaches

Our experimental results (Section 7.1) suggest that CP is good at finding a feasible set of weights, while MIP is good at optimizing them towards solutions that generalize better. This motivates our hybrid methods that find a first feasible solution using CP and then use a MIP model to optimize. We use a CP model without objective function, $\texttt{CP}_f$, since it finds feasible solutions on a larger number of instances than $\texttt{CP}_w$ and $\texttt{CP}_m$ (Section 7.2).

We propose two alternatives to incorporate the CP solution into the MIP models. Our first hybrid model, HW, uses the CP solution as a warm-start for either $\texttt{MIP}_w$ or $\texttt{MIP}_m$. The second hybrid variant, HA, fixes the activations of all the neurons in the MIP model and searches only over the weights. As a result,

all the big-M constraints and variables $c^k_{i\ell j}$ are removed, albeit at the cost of potentially pruning optimal solutions.

Given a feasible set of weights $\hat{w}_{i\ell j}$ returned by $\texttt{CP}_f$, HA computes the neuron activations for a training example $k \in T$ as $\hat{n}^k_{0i} = x^k_i$ for the input layer and $\hat{n}^k_{\ell i} = 2\left(\sum_{i \in N_{\ell-1}} \hat{n}^k_{(\ell-1)i} \hat{w}_{i\ell j} \geq 0\right) - 1$ for $\ell \in \mathcal{L}$. Then, the fixed activation models for min-weight $\texttt{HA}_w$ and max-margin $\texttt{HA}_m$ are as follows.

$$\min \sum_{\ell \in \mathcal{L}} \sum_{i \in N_{\ell-1}} \sum_{j \in N_\ell} v_{i\ell j} \qquad (\texttt{HA}_w)$$

$$\text{s.t. } (14), (15), (18)$$

$$\sum_{i \in N_{\ell-1}} w_{i\ell j} \cdot \hat{n}^k_{(\ell-1)i} \geq 0 \qquad \forall \ell \in \mathcal{L}, j \in N_\ell, k \in T : \hat{n}^k_{\ell j} = 1 \qquad (24)$$

$$\sum_{i \in N_{\ell-1}} w_{i\ell j} \cdot \hat{n}^k_{(\ell-1)i} \leq -\epsilon \qquad \forall \ell \in \mathcal{L}, j \in N_\ell, k \in T : \hat{n}^k_{\ell j} = -1 \qquad (25)$$

$$\max \sum_{\ell \in \mathcal{L}} \sum_{j \in N_\ell} m_{\ell j} \qquad (\texttt{HA}_m)$$

$$\text{s.t. } (15), (23)$$

$$\sum_{i \in N_{\ell-1}} w_{i\ell j} \cdot \hat{n}^k_{(\ell-1)i} \geq m_{\ell j} \qquad \forall \ell \in \mathcal{L}, j \in N_\ell, k \in T : \hat{n}^k_{\ell j} = 1 \qquad (26)$$

$$\sum_{i \in N_{\ell-1}} w_{i\ell j} \cdot \hat{n}^k_{(\ell-1)i} \leq -\epsilon - m_{\ell j} \qquad \forall \ell \in \mathcal{L}, j \in N_\ell, k \in T : \hat{n}^k_{\ell j} = -1 \qquad (27)$$

Hybrid methods are not necessary when the BNN has no hidden layers; in such scenarios, the implication constraints (7)–(8) and (21)–(22) are not needed and, as a result, the HA models reduce to our MIP models.

6 Gradient Descent Baselines

Current methods to train BNNs follow Hubara et al.'s GD-based algorithm [9] described in Section 3. This algorithm is a highly optimized local search method that starts from a random weight assignment and locally changes the weights towards minimizing a *Square Hinge loss* function. The Square Hinge loss function quadratically penalizes the errors on the training set. The most relevant hyper-parameter is the *learning rate* that defines how much each weight is updated in every step.

Hubara et al.'s approach learns BNNs only with -1 and $+1$ weights, whereas our models also allow for zero-value weights. To make a fair comparison, we also extended Hubara et al.'s approach to work with zero-value weights. Instead of learning one binary weight per connection we learn two, w^1_b and w^2_b. The final weight for the connection is the average between those two values, i.e, $w_b = (w^1_b + w^2_b)/2 \in \{-1, 0, 1\}$. Our experiments report the performance of both the original approach $\texttt{GD}_b$ and our extension $\texttt{GD}_t$.

Fig. 2: The first 10 examples from the MNIST dataset.

7 Experimental Evaluation for Few-Shot Learning

We tested our models over subsets of the MNIST dataset [19], which consists of $70,000$ labeled images of handwritten digits. Each image has 28×28 gray-scale pixels with values between 0 to 255. Every example has a label representing the digit that appears on the input image, i.e., its class. Figure 2 shows 10 examples from the MNIST training set.

To emulate the conditions of a few-shot learning scenario, we limited the training set size to a range varying from 1 to 10 examples per class. We sampled 10 problem instances for each class and trained BNNs with $28 \times 28 = 784$ input neurons and 10 output neurons (one per class). If the image label is i, then the ith output neuron should be active ($y_i = 1$) and the rest inactive ($y_j = -1$ for all $j \neq i$). Each BNN has 0, 1, or 2 hidden layers with 16 neurons each.

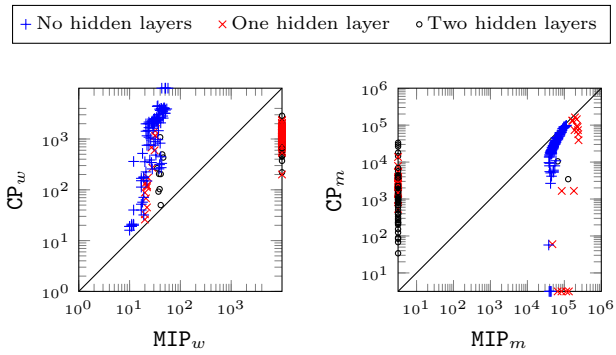
We compare our models using three metrics. The first two metrics correspond to the number of instances solved (i.e., finding a weight assignment that fits the training data) and the quality of those solutions w.r.t. the objective functions. The third metric compares the test performance over the $10,000$ test instances from MNIST. We use the *all-good* metric that evaluates the percentage of instances where the value of the 10 output neurons is correct. As such, the expected performance of a BNN with random weights is 0.098%.

Approaches. We use Gurobi 8.1 [8] to solve the MIP models and IBM ILOG CP Optimizer 12.8 [10] for the CP models. For MIP, the implications are formulated using Gurobi's special construct. The GD baselines were solved using Tensorflow 1.9.0 and Adam optimizer [16]. We evaluated the following approaches:[5]

- $\texttt{CP}_w$ and $\texttt{CP}_m$: min-weight and max-margin CP models, respectively.
- $\texttt{MIP}_w$ and $\texttt{MIP}_m$: min-weight and max-margin MIP models, respectively.
- $\texttt{HW}_w$ and $\texttt{HW}_m$: min-weight and max-margin warm-start hybrid models.
- $\texttt{HA}_w$ and $\texttt{HA}_m$: min-weight and max-margin fixed-activation hybrid models.
- $\texttt{GD}_b$ and $\texttt{GD}_t$: Hubara et al.'s approach [9] and our extension for zero-weights.

As the GD baselines find different solutions depending on their starting point and learning rate, we tested four common learning rates $(10^{-3}, \ldots, 10^{-6})$ starting from 5 independently sampled BNNs for each problem instance. We defined the performance of each learning rate to be the average performance across its five starting points. Our experimental results report the performance of the *best* learning rate for each problem. This is an upper bound on the GD performance

[5] Our source code is publicly available at `https://bitbucket.org/RToroIcarte/bnn`.



(a) Min-weight optimization (b) Max-margin optimization

Fig. 3: Solution quality comparison between CP and MIP.

that assumes the existence of an oracle that can predict the best learning rate for each sampled training instance and BNN architecture.

Each approach was run with a 2-hour time limit using one thread on an Intel Xeon E5-2680 2.70GHz processor with 96GB of RAM. This time limit is long enough for $\texttt{GD}_b$ and $\texttt{GD}_t$ to converge in most of our experiments.

Data preprocessing. An input neuron is considered *dead* if its value is the same in the entire training set. As those neurons add no new information to discriminate the correct output for a given input, they can be removed without losing correctness. We exploit this structure in our models (and baselines) by fixing the value of every weight connected to a dead input neuron to zero.

7.1 Solution Quality Comparison Between MIP and CP

We now compare the efficiency of our monolithic models for finding high-quality BNNs that fit the training data. Figure 3 compares the quality of the solutions found by the MIP and CP models for the min-weight and max-margin objectives. A point (x, y) in the plots corresponds to a single instance where x and y represent the objective value obtained using MIP and CP, respectively. Points that appear along the vertical (resp. horizontal) axes correspond to instances where MIP (resp. CP) timed out before finding any feasible solution. For the min-weight objective, points above the diagonal represent instances where the MIP model found better solutions. The inverse is true for the max-margin graph.

The results show that MIP struggled to find feasible solutions when using one and two hidden layers. This is mainly explained by the large number of big-M constraints and variables that both $\texttt{MIP}_w$ and $\texttt{MIP}_m$ have in those cases. In contrast, CP found feasible solutions for most of the problem instances.

When both methods found feasible solutions, the graphs suggest that MIP was better at finding high-quality solutions. With both objectives, MIP consistently found equal or better quality solutions than CP. In fact, MIP found

Table 1: Number of instances where a feasible solution was found.

	One hidden layer										Two hidden layers										
$\|\mathcal{T}\|$	10	20	30	40	50	60	70	80	90	100	10	20	30	40	50	60	70	80	90	100	**Total**
$\texttt{GD}_b$	9.4	0.2	0	0	0	0	0	0	0	0	5.6	0	0	0	0	0	0	0	0	0	15.2
$\texttt{GD}_t$	9.6	5.6	0.4	0	0	0	0	0	0	0	9.2	8.4	5.2	6.2	4.2	2.2	0	0	0	0	51
$\texttt{MIP}_m$	**10**	3	2	1	0	1	0	0	0	0	2	0	0	0	0	0	0	0	0	0	19
$\texttt{MIP}_w$	**10**	7	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	0	26
$\texttt{CP}_m$	**10**	6	6	3	3	3	0	0	0	0	**10**	**10**	**10**	**10**	**10**	9	6	5	2	0	103
$\texttt{CP}_w$	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**8**	**10**	**10**	**10**	8	4	7	2	0	0	0	149
$\texttt{CP}_f$	**10**	**10**	9	8	8	7	3	3	1	0	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**8**	**6**	**153**

proven optimal solutions for 68 out of 300 instances when minimizing weights and 15 out of 300 when maximizing margins, while CP never found and proved optimal solutions.

7.2 Comparison Between Hybrid Methods

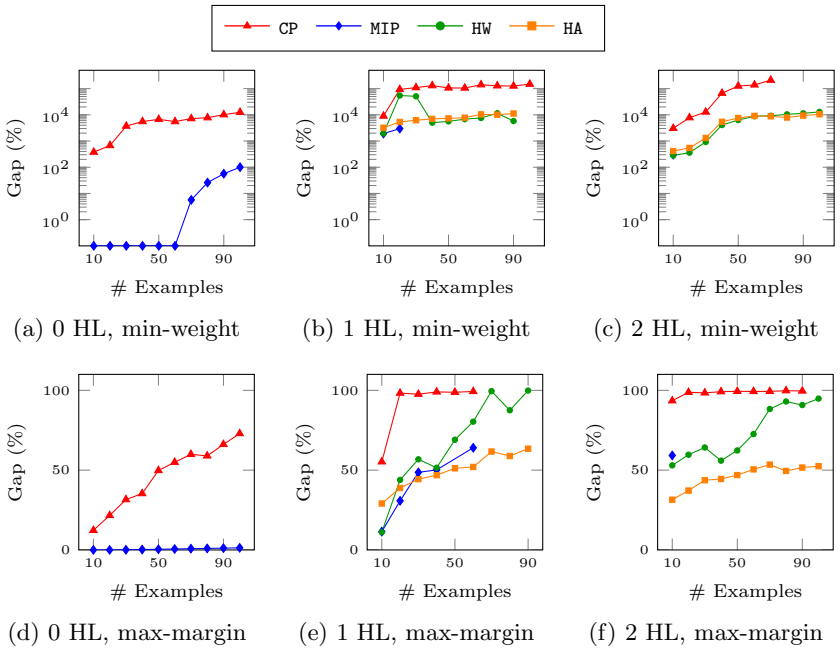
When hidden layers are used, our hybrid methods find a first feasible solution using a CP model without an objective function, $\texttt{CP}_f$, and give it to a MIP model to optimize. To find feasible solutions, we could have instead used MIP, GD, $\texttt{CP}_w$, or $\texttt{CP}_m$. However, $\texttt{CP}_f$ tends to finds more feasible solutions than the other methods. This is well-supported by Table 1, which shows the number of instances where a feasible solution was found (for each method) in under 2 hours.

To compare the solution quality of our model-based approaches, we analyze the optimality gaps across different network architectures and training examples. The gap computation uses the best dual bound found by any approach. Figure 4 shows the average optimality gaps obtained for the min-weight and max-margin criteria using the monolithic and hybrid methods. We omit the gap lines for HA and HW for the experiments with no hidden layer since our hybrid methods are not needed in this case (see Section 5).

These results suggest that the hybrid methods exhibit the best characteristics of the CP and MIP models. HW and HA scaled to larger training sets and network architectures in a manner similar to CP—significantly outperforming MIP in this metric—while obtaining high quality solutions that are comparable to those produced by MIP. In addition, HA consistently outperformed HW when maximizing the margins and found similar quality solutions for the min-weight criteria.

7.3 Test Performance Comparison with GD

Figure 5 compares the test performance of our methods and the best performing GD baseline. A data point represents the performance of a BNN for each training set and network architecture. Points below the diagonal represent instances where our approaches outperformed GD. These results show that MIP outperformed GD methods when it found a solution, and that the hybrid methods found many more solutions than MIP while maintaining a similar test performance profile. In particular, $\texttt{HA}_m$ has a remarkable performance in comparison



(a) 0 HL, min-weight (b) 1 HL, min-weight (c) 2 HL, min-weight

(d) 0 HL, max-margin (e) 1 HL, max-margin (f) 2 HL, max-margin

Fig. 4: Optimality gap comparison with different number of hidden layers (HL) and size of the training set (# Examples).

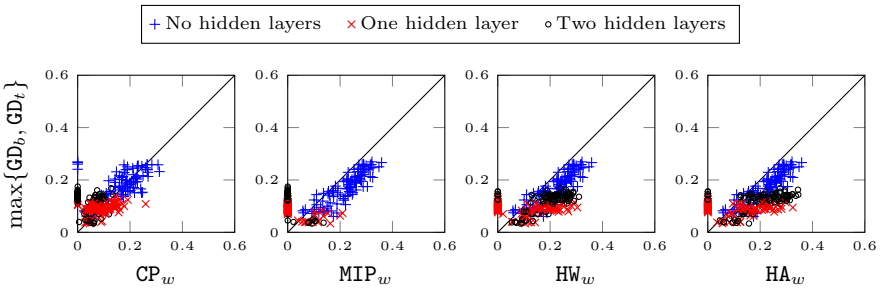
with GD in these experiments. For instance, with 2 hidden layers and 100 training examples, $\texttt{HA}_m$ correctly classifies up to $5,612$ of $10,000$ unseen examples while GD predicted the true class in at most $1,563$ cases. Note that a BNN with random weights is expected to correctly predict less than 10 examples.

To better represent these results, Figure 6 shows the number of instances where model-based approaches have a strictly better test performance than the best GD baseline. When limited data was used, the hybrid approaches consistently outperformed GD. However, as the training sets get larger, some of our models timed out before finding feasible solutions. In contrast, GD always returned a solution. Such solution might not fit all the training data but it can still be evaluated on the test set. Hence, GD is superior with large training sets.

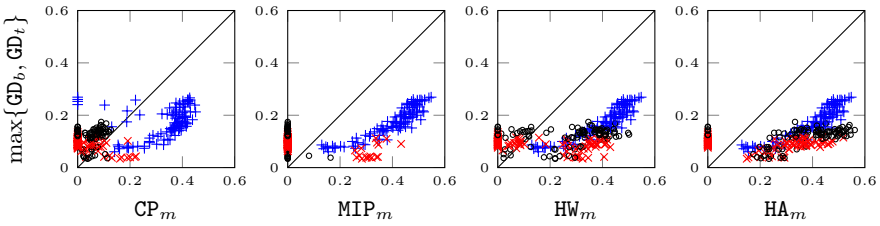
It is equally important to consider by how much the solutions found by our models outperform the solutions found by GD. Figure 7 displays the average test performance across the instances solved by each approach. Under this metric, the clear winner is $\texttt{HA}_m$ (which reduces to $\texttt{MIP}_m$ when no hidden layers are used) as it largely outperformed GD and CP while scaling better than MIP.

7.4 Discussion

Our experiments demonstrate the merits of model-based approaches—in particular, MIP and CP—to train BNNs. When data is scarce, these methods can



(a) Min-weight test performance comparison with GD



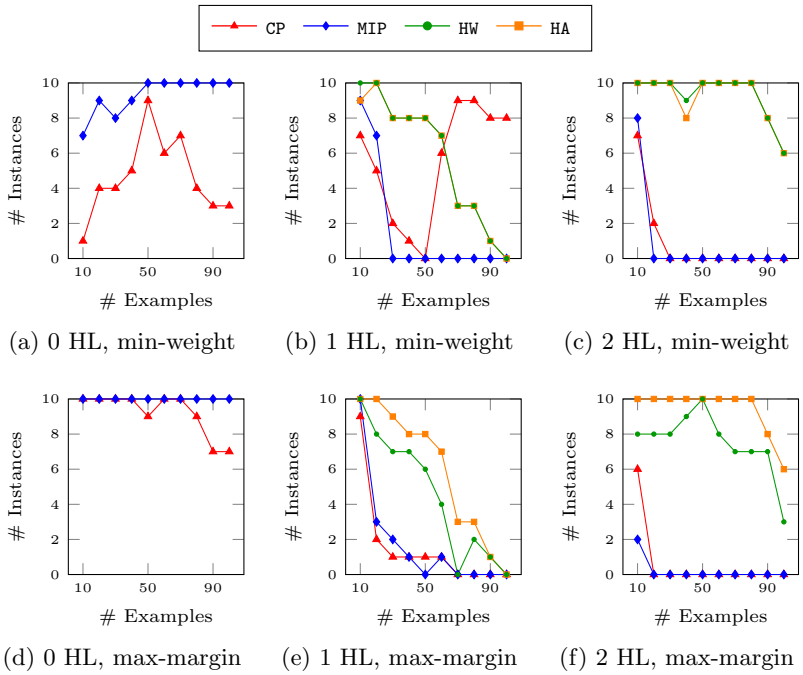
(b) Max-margin test performance comparison with GD

Fig. 5: Test performance comparison between the best GD and model-based methods with different number of hidden layers (HL) and optimization criteria.

find solutions that generalize better than the solutions found by GD. This is a notable result that opens many opportunities for future work. In particular, there are three interesting questions that arise from our experimental evaluation.

What are the advantages and limitations of model-based approaches? The main advantage of training BNNs using model-based approaches is in finding solutions that generalize better using fewer examples. Consider the results on Figure 7(e) and 7(f). They show that our hybrid models need only 10 examples to find solutions that generalize better than the ones found by GD using 100 examples. That being said, their main limitation is scalability. We expect that more sophisticated model-based approaches, such as decompositions and specialized CP propagators, will push the boundary of problems that can be solved. We also believe that model-based approaches will become a new tool for ML researchers, as they allow for principled empirical comparisons of generalization criteria based on provable bounds.

Are min-weight and max-margin the best proxies for generalization? Our results suggest that both min-weight and max-margin are good proxies for generalization. In fact, given two BNNs that perfectly fit the training data, our models can accurately predict which one generalizes better. Through a pairwise comparison of all the perfect-fit BNNs generated for each instance in our exper-



(a) 0 HL, min-weight (b) 1 HL, min-weight (c) 2 HL, min-weight

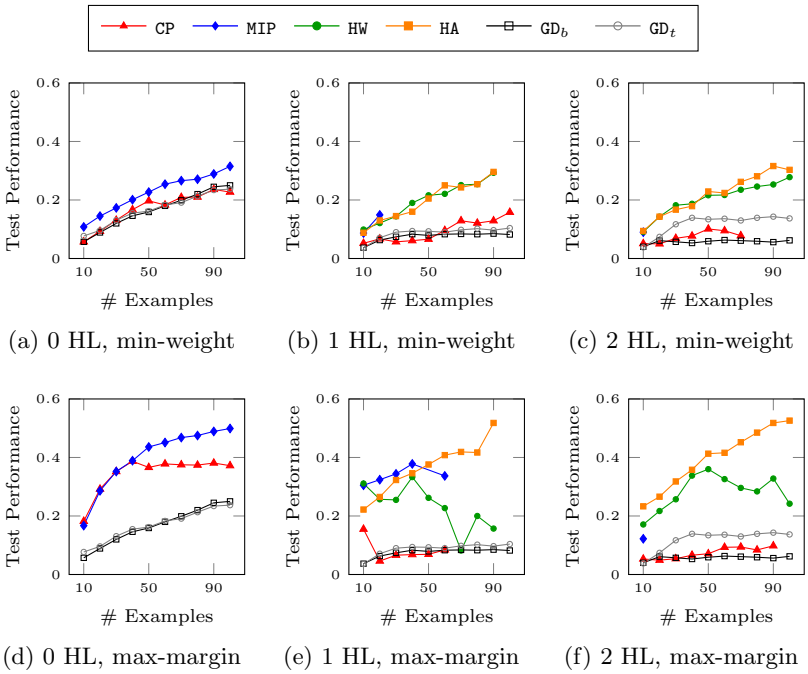
(d) 0 HL, max-margin (e) 1 HL, max-margin (f) 2 HL, max-margin

Fig. 6: Number of instances where model-based approaches have better test performance than $\max\{\texttt{GD}_b, \texttt{GD}_t\}$.

iments, we saw that the BNN with bigger margin generalized better over 85% of the time. The min-weight criteria is not as good at predicting generalization, but still does a reasonable job: over 79% of the time, the BNN with fewer nonzero weights generalized better. However, it would be very surprising if there are no other criteria that could better predict generalization. Looking for such criteria is a promising future work direction.

Why are deep BNNs not generalizing better than shallow ones? A major insight from the deep learning literature is that adding hidden layers improves generalization [28]. Surprisingly, this was not the case in our experiments. A possible explanation is that our training sets are not big enough to justify the use of hidden layers. This a reasonable hypothesis, especially considering that the test performance of GD methods also decreased when adding more hidden layers (see Figure 7). However, it does not explain why adding hidden layers improves generalization when using 10 training examples for $\texttt{HW}_m$ and $\texttt{HA}_m$.

Another possible explanation is that we are not finding close-to-optimal solutions when using hidden layers (see Figure 4). Hence, while the test performance reaches its full potential for the case with no hidden layers, there is room for improvement for BNNs with hidden layers.



(a) 0 HL, min-weight (b) 1 HL, min-weight (c) 2 HL, min-weight

(d) 0 HL, max-margin (e) 1 HL, max-margin (f) 2 HL, max-margin

Fig. 7: Test performance comparison for all our methods in BNNs with different number of hidden layers (HL) using the two optimization criteria.

8 Concluding Remarks

Our work examines the use of MIP and CP to train BNNs. We formulate the training problem as finding a BNN that perfectly fits the training set while optimizing two proxies for generalizability. When solving this problem, we note that CP is good at finding feasible solutions and MIP is good at optimizing them. Hence, we propose two CP/MIP hybrids that exploit the strengths of CP and MIP. With limited training data, our hybrid approaches found BNNs that generalized better than the ones found by GD. In contrast, GD scaled better, making it more appealing when large training sets are available.

This work opens many opportunities for future work at the intersection between ML and OR. From an ML perspective, model-based approaches allow for principled empirical comparisons between proxies for generalization and seem effective for few-shot learning. From an OR perspective, training BNNs is a challenging combinatorial optimization problem with interesting structure. We believe that exploiting such structure via decompositions or specialized CP propagators presents a promising direction for future work.

Acknowledgements

We would like to thank Toryn Klassen, Maayan Shvo, and Ethan Waldie for their help running experiments and Kyle Booth, Arik Senderovich, and the anonymous reviewers for helpful comments. We gratefully acknowledge funding from CONICYT (Becas Chile), NSERC, and Microsoft Research.

References

1. Anderson, R., Huchette, J., Tjandraatmadja, C., Vielma, J.P.: Strong mixed-integer programming formulations for trained neural networks. In: Proceedings of the International Conference on Integer Programming and Combinatorial Optimization (ICPO). pp. 27–42. Springer (2019)
2. Cheng, C.H., Nührenberg, G., Huang, C.H., Ruess, H.: Verification of binarized neural networks via inter-neuron factoring. In: Proceedings of the 10th Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE). pp. 279–290. Springer (2018)
3. Ching, T., Himmelstein, D.S., Beaulieu-Jones, B.K., Kalinin, A.A., Do, B.T., Way, G.P., Ferrero, E., Agapow, P.M., Zietz, M., Hoffman, M.M., et al.: Opportunities and obstacles for deep learning in biology and medicine. Journal of The Royal Society Interface **15**(141), 20170387 (2018)
4. Domingos, P.: A few useful things to know about machine learning. Communications of the ACM **55**(10), 78–87 (2012)
5. Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. Constraints **23**, 296–309 (2018)
6. Gambella, C., Ghaddar, B., Naoum-Sawaya, J.: Optimization models for machine learning: A survey. arXiv preprint arXiv:1901.05331 (2019)
7. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)
8. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2018), `http://www.gurobi.com`
9. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks. In: Proceedings of the 29th Conference on Advances in Neural Information Processing Systems (NIPS). pp. 4107–4115 (2016)
10. IBM: ILOG CP Optimizer 12.8 Manual (2018)
11. Jiang, Y., Krishnan, D., Mobahi, H., Bengio, S.: Predicting the generalization gap in deep networks with margin distributions. In: ICLR2019 (2019)
12. Kawaguchi, K., Kaelbling, L.P., Bengio, Y.: Generalization in deep learning. arXiv preprint arXiv:1710.05468 (2017)
13. Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P.: On large-batch training for deep learning: Generalization gap and sharp minima. Proceedings of the 5th International Conference on Learning Representations (ICLR) (2017)
14. Khalil, E.B., Dilkina, B.: Training binary neural networks with combinatorial algorithms. Extended abstract at the 15th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR) (2018)
15. Khalil, E.B., Gupta, A., Dilkina, B.: Combinatorial attacks on binarized neural networks. In: Proceedings of the 7th International Conference on Learning Representations (ICLR) (2019)

16. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. Proceedings of the 3rd International Conference on Learning Representations (ICLR) (2015)
17. Lahoud, F., Achanta, R., Márquez-Neila, P., Süsstrunk, S.: Self-binarizing networks. arXiv preprint arXiv:1902.00730 (2019)
18. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
19. LeCun, Y., Cortes, C., Burges, C.J.: The MNIST database of handwritten digits. URL http://yann.lecun.com/exdb/mnist (1998)
20. Li, F., Zhang, B., Liu, B.: Ternary weight networks. arXiv preprint arXiv:1605.04711 (2016)
21. Miotto, R., Wang, F., Wang, S., Jiang, X., Dudley, J.T.: Deep learning for healthcare: review, opportunities and challenges. Briefings in bioinformatics **19**(6), 1236–1246 (2017)
22. Mishra, A., Marr, D.: Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. Proceedings of the 6th International Conference on Learning Representations (ICLR) (2018)
23. Moody, J.E.: The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In: Proceedings of the 4th Conference on Advances in Neural Information Processing Systems (NIPS). pp. 847–854 (1991)
24. Narodytska, N.: Formal analysis of deep binarized neural networks. In: Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI). pp. 5692–5696 (2018)
25. Neyshabur, B., Bhojanapalli, S., McAllester, D., Srebro, N.: Exploring generalization in deep learning. In: Proceedings of the 30th Conference on Advances in Neural Information Processing Systems (NIPS). pp. 5947–5956 (2017)
26. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet classification using binary convolutional neural networks. In: Proceedings of the 14th European Conference on Computer Vision (ECCV). pp. 525–542 (2016)
27. Schmidhuber, J.: Deep learning in neural networks: An overview. Neural networks **61**, 85–117 (2015)
28. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. Proceedings of the 3rd International Conference on Learning Representations (ICLR) (2015)
29. Suykens, J.A., Vandewalle, J.: Least squares support vector machine classifiers. Neural processing letters **9**(3), 293–300 (1999)
30. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. Proceedings of the 7th International Conference on Learning Representations (ICLR) (2019)
31. Umuroglu, Y., Fraser, N.J., Gambardella, G., Blott, M., Leong, P., Jahre, M., Vissers, K.: FINN: A framework for fast, scalable binarized neural network inference. In: Proceedings of the 25th International Symposium on Field-Programmable Gate Arrays (FPGA). pp. 65–74 (2017)
32. Vanschoren, J.: Meta-learning: A survey. arXiv preprint arXiv:1810.03548 (2018)
33. Wan, D., Shen, F., Liu, L., Zhu, F., Qin, J., Shao, L., Tao Shen, H.: TBN: Convolutional neural network with ternary inputs and binary weights. In: Proceedings of the 15th European Conference on Computer Vision (ECCV). pp. 315–332 (2018)