Multiobjective Linear Ensembles for Robust and Sparse Training of Few-Bit Neural Networks

Bernardelli, Ambrogio Maria; Gualandi, Stefano; Milanesi, Simone; Lau, Hoong Chuin; Yorke-Smith, Neil

DOI
10.1287/ijoc.2023.0281

Publication date
2025

Published in
INFORMS Journal on Computing

https://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Articles in Advance, pp. 1–21
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

Multiobjective Linear Ensembles for Robust and Sparse Training of Few-Bit Neural Networks

Ambrogio Maria Bernardelli,[a,*] Stefano Gualandi,[a] Simone Milanesi,[a] Hoong Chuin Lau,[b] Neil Yorke-Smith[c]

[a] Department of Mathematics, University of Pavia, 27100 Pavia, Italy; [b] School of Computing and Information Systems, Singapore Management University, Singapore 178902, Singapore; [c] Socio-Technical Algorithmic Research (STAR) Laboratory, Delft University of Technology, 2600 GA Delft, Netherlands

*Corresponding author

Contact: ambrogiomaria.bernardelli01@universitadipavia.it, https://orcid.org/0000-0002-2328-7062 (AMB); stefano.gualandi@unipv.it, https://orcid.org/0000-0002-2111-3528 (SG); simone.milanesi01@universitadipavia.it, https://orcid.org/0000-0002-6314-1965 (SM); hclau@smu.edu.sg, https://orcid.org/0000-0002-5326-411X (HCL); n.yorke-smith@tudelft.nl, https://orcid.org/0000-0002-1814-3515 (NY-S)

Received: August 9, 2023
Revised: March 30, 2024; June 30, 2024
Accepted: July 2, 2024
Published Online in Articles in Advance: September 13, 2024

https://doi.org/10.1287/ijoc.2023.0281

Abstract. Training neural networks (NNs) using combinatorial optimization solvers has gained attention in recent years. In low-data settings, the use of state-of-the-art mixed integer linear programming solvers, for instance, has the potential to exactly train an NN while avoiding computing-intensive training and hyperparameter tuning and simultaneously training and sparsifying the network. We study the case of few-bit discrete-valued neural networks, both binarized neural networks (BNNs) whose values are restricted to ± 1 and integer-valued neural networks (INNs) whose values lie in the range $\{-P, \ldots, P\}$. Few-bit NNs receive increasing recognition because of their lightweight architecture and ability to run on low-power devices: for example, being implemented using Boolean operations. This paper proposes new methods to improve the training of BNNs and INNs. Our contribution is a multiobjective ensemble approach based on training a single NN for each possible pair of classes and applying a majority voting scheme to predict the final output. Our approach results in the training of robust sparsified networks whose output is not affected by small perturbations on the input and whose number of active weights is as small as possible. We empirically compare this *BeMi* approach with the current state of the art in solver-based NN training and with traditional gradient-based training, focusing on BNN learning in few-shot contexts. We compare the benefits and drawbacks of INNs versus BNNs, bringing new light to the distribution of weights over the $\{-P, \ldots, P\}$ interval. Finally, we compare multiobjective versus single-objective training of INNs, showing that robustness and network simplicity can be acquired simultaneously, thus obtaining better test performances. Although the previous state-of-the-art approaches achieve an average accuracy of 51.1% on the Modified National Institute of Standards and Technology data set, the *BeMi* ensemble approach achieves an average accuracy of 68.4% when trained with 10 images per class and 81.8% when trained with 40 images per class while having up to 75.3% NN links removed.

History: Accepted by Andrea Lodi, Area Editor for Design & Analysis of Algorithms—Discrete.
Funding: This research was partially supported by the European Union Horizon 2020 Research and Innovation Programme [Grant 952215]. The work of A. M. Bernardelli is supported by a PhD scholarship funded under the "Programma Operativo Nazionale Ricerca e Innovazione" 2014–2020.
Supplemental Material: The software that supports the findings of this study is available within the paper as well as from the IJOC GitHub software repository (https://github.com/INFORMSJoC/2023.0281). The complete IJOC Software and Data Repository is available at https://informsjoc.github.io/.

Keywords: binarized neural networks • integer neural networks • mixed-integer linear programming • structured ensemble • few-shot learning • sparsity • multiobjective optimization

1. Introduction

State-of-the-art deep neural networks (NNs) contain a huge number of neurons organized in many layers, and they require an immense amount of data for training (LeCun et al. 2015). The training process is computationally demanding and is typically performed by stochastic gradient descent algorithms running on large Graphics Processing Unit (GPU)-based or even tensor processing unit (TPU)-based clusters. Whenever the trained (deep) neural network contains many neurons, the network deployment is also computationally demanding. However, in some real-life applications, extensive GPU-based training might be infeasible, training data might be scarce with

only a few data points per class, or the hardware using the NN at inference time might have a limited computational power as, for instance, whenever the NN is executed on an industrial embedded system (Blott et al. 2019).

Binarized neural networks (BNNs) were introduced in Hubara et al. (2016) as a response to the challenge of running NNs on low-power devices. BNNs contain only binary weights and binary activation functions, and hence, they can be implemented using only efficient bit-wise operations. However, the training of BNNs raises interesting challenges for gradient-based approaches because of their combinatorial structure. In previous works (Toro Icarte et al. 2019), it was shown that the training of a BNN can be performed by using combinatorial optimization solvers; a hybrid constraint programming (CP) and mixed integer programming (MIP) approach outperformed the stochastic gradient approach proposed by Hubara et al. (2016) if restricted to a few-shot learning context (Vanschoren 2019). The work of Toro Icarte et al. (2019) has been furthered by a number of authors more recently as we survey in the next section.

Indeed, combinatorial approaches (principally, MIP) for training neural networks, both discrete and continuous, have been employed in the literature as demonstrated in subsequent works (e.g., Kurtz and Bah 2021, Patil and Mintz 2022). MIP optimization has been explored in the machine learning community, such as in Li et al. (2020), Huang et al. (2023), and Ye et al. (2023) for instance. Various architectures and activation functions have been utilized in these studies. Solver-based training has the advantage that, in principle, the optimal NN weights can be found for the training data and that network optimization (e.g., pruning) or adversarial hardening can be performed.

The main challenge in training an NN by an exact MIP-based approach is the limited amount of training data that can be used because otherwise, the size of the optimization model explodes. In the recent work of Thorbjarnarson and Yorke-Smith (2023), the combinatorial training idea was further extended to integer-valued neural networks (INNs). Exploiting the flexibility of MIP solvers, the authors were able to (i) minimize the number of neurons during training and (ii) increase the number of data points used during training by introducing an MIP batch training method.

We remark that training an NN with an MIP-based approach is more challenging than solving a verification problem, such as in Fischetti and Jo (2018) and Anderson et al. (2020), even if the structure of the nonlinear constraints modeling the activation functions is similar. In NNs verification (Khalil et al. 2019), the weights are given as input, whereas in MIP-based training, the weights are the *decision variables* that must be computed.

We note several lines of work aimed at producing compact and simple NNs that maintain acceptable accuracy (e.g., in terms of parameter pruning (Serra et al. 2020, Yu et al. 2022, Cai et al. 2023, ElAraby et al. 2023), loss function improvement (Tang et al. 2017), gradient approximation (Sakr et al. 2018), and network topology structure (Lin et al. 2017)).

In the context of MIP-based training and optimization of NNs, this paper proposes new methods to improve the training of BNNs and INNs. In summary, our contributions are (i) the formulation of an MILP model with a multiobjective target that consists of already existing single-objective steps in a lexicographic order; (ii) the implementation of an ensemble of few-bits NNs, in which each of them is specialized in a specific classification task; and (iii) the proposal of a voting scheme inspired by a one-versus-one (OVO) strategy, tailored specifically for the constructed ensemble of NNs. Our computational results using the Modified National Institute of Standards and Technology (MNIST) and the Fashion-MNIST data set show that the *BeMi*[1] ensemble permits us to use for training up to 40 data points per class thanks to the fact that the OVO strategy results in smaller MILPs, reaching an average accuracy of 81.8% for MNIST and 70.7% for Fashion-MNIST. In addition, thanks to the multiobjective function that minimizes the number of links (i.e., the connections between different neurons), up to 75% of weights are set to zero for MNIST, and up to 48% of weights are set to zero for Fashion-MNIST. We also perform additional experiments on the Heart Disease data set, reaching an average accuracy of 78.5%. A preliminary report of this work appeared at the Learning and Intelligent Optimization 2023 conference (Bernardelli et al. 2023). This paper better motivates and situates our approach in the state of the art, develops our ensemble approach for INNs (not only BNNs), presents more extensive and improved empirical results, and analyzes the distribution of INN weights.

1.1. Outline

The remainder of this paper is as follows. Section 2 situates our work in the literature. Section 3 introduces the notation and defines the problem of training a single INN with the existing MIP-based methods. Section 4 presents the *BeMi* ensemble, the majority voting scheme, and the improved MILP model to train a single INN. Section 5 presents the computational results on the MNIST, Fashion-MNIST, and Heart Disease data sets. Finally, Section 6 concludes the paper with a perspective on future work.

2. Related Works

Recently, there has been growing research interest in studying the impact of machine learning on improving traditional operations research methods (e.g., see Bengio et al. 2021 and Cappart et al. 2023), in designing integrated

predictive and modeling frameworks as in Bergman et al. (2022), or in embedding a pretrained machine learning model into an optimization problem (e.g., see Lombardi and Milano 2018, Mistry et al. 2021, and Tsay et al. 2021). In this work, we take a different perspective, and we study how an exact MILP solver can be used for training machine learning models: more specifically, to train (binary or integer) neural networks. In the following paragraphs, we first review two recent applications to MILP solvers in the context of neural networks (i.e., weight pruning and NN verification), and later, we review the few works that have tackled the challenge of training an NN using an exact solver.

2.1. MIP-Based Neural Networks Pruning

Recent works have shown interesting results on *pruning* a trained neural network using an optimization approach based on the use of MIP solvers (Serra et al. 2020, Good et al. 2022, Yu et al. 2022, Cai et al. 2023, ElAraby et al. 2023). When pruning a trained neural network, the weights are fixed, and the optimization variables represent the decision to keep or remove an existing weight different from zero.

2.2. MIP-Based Neural Networks Verification

Another successful application of exact solvers in the context of neural networks is for tackling the verification problem: that is, to verify under which conditions the accuracy of a given trained neural network does not deteriorate. In other words, in NN verification, the optimization problem consist of finding adversarial examples using a minimal distortion of the input data. The use of MIP solvers for this application was pioneered in Tjeng et al. (2018) and later studied in several papers, such as Fischetti and Jo (2018), Tjeng et al. (2018), Anderson et al. (2020), Botoeva et al. (2020), and Tjandraatmadja et al. (2020). For a broader discussion of the use of the polyhedral approach to verification, see Huchette et al. (2023, section 4). In NN verification, the weights of the NN are given as input, and the optimization problem consists of assessing how much the input can change without compromising the output of the network. Indeed, verification is also a different optimization problem than the exact training that we discuss in the following sections.

2.3. Exact Training of Neural Networks

The utilization of MIP approaches for training neural networks has already been explored in the literature, primarily in the context of few-shot learning and NNs with low-bit parameters (Toro Icarte et al. 2019, Thorbjarnarson and Yorke-Smith 2023). One of the main advantages of these approaches is the ability to simultaneously train and optimize the network architecture. Although the modeling that defines the structure of the neural network is rigid and heavily relies on the discrete nature of the parameters, the choice of the objective function provides more flexibility and allows for the optimization of various network characteristics. For instance, it enables the minimization of the number of connections in a fixed architecture network, thereby promoting lightweight architectures.

From now on, by INN, we indicate a general NN whose weights take value in the set $\{-P, \dots, P\}$, where integer $P \geq 1$. Notice that this choice includes the special case of BNNs ($p = 1$). When referring to an INN with $p > 1$, we will write *nontrivial* INN.

By incorporating the power of both CP and MIP, the study of Toro Icarte et al. (2019) showcased the effectiveness of leveraging combinatorial optimization methods for training BNNs in a few-shot learning context. Two types of objective functions were selected to drive the optimization process. The first objective function aimed at promoting a lightweight architecture by minimizing the number of nonzero weights. This objective sought to reduce the overall complexity of the neural network, allowing for efficient computation and resource utilization. Note that with this choice, pruning is not necessary; the possibility of setting a weight to zero is equivalent to removing the corresponding weight (i.e., connection). In contrast, the second objective function focused on enhancing the robustness of the network, particularly in the face of potential noise in the input data. Robustness here refers to the ability of the network to maintain stable and reliable performance even when the input exhibits variations or disturbances. The research of Thorbjarnarson and Yorke-Smith (2023) extends the single-objective approach of Toro Icarte et al. (2019) to the broader class of integer-valued neural networks. In addition to leveraging the objectives of architectural lightness and robustness, a novel objective aimed at maximizing the number of correctly classified training data instances is introduced. It is worth noting that this type of objective shares some similarities with the goals pursued by gradient descent methods, albeit with a distinct formulation. An interesting observation is that this objective formulation remains feasible in practice, ensuring that a valid solution can be obtained. We remark that both papers propose single-objective models that involve training a single neural network to approximate a multiclassification function.

Ensembles of neural networks are well known to yield more stable predictions and demonstrate superior generalizability compared with single neural network models (Wang et al. 2023). In this paper, we aim to redefine the

concept of structured ensemble by composing our ensemble of several networks, each specialized in a distinct task. Given a classification task over k classes, the main idea is to train $\frac{k(k-1)}{2}$ INNs, where every single network learns to discriminate only between a given pair of classes. When a new data point (e.g., a new image) must be classified, it is first fed into the $\frac{k(k-1)}{2}$ trained INNs, and later, using a Condorcet-inspired majority voting scheme (Young 1988), the most frequent class is predicted as output. This method is similar to and generalizes the support vector machine and OVO approach (Bishop and Nasrabadi 2006), whereas it has not yet been applied within the context of neural networks to the best of our knowledge.

For training every single INN, our approach extends the methods introduced in Toro Icarte et al. (2019) and Thorbjarnarson and Yorke-Smith (2023), as described above.

3. Few-Bit Neural Networks

In this section, we formally define a binarized neural network and an integer-valued neural network using the notation as in Toro Icarte et al. (2019) and Thorbjarnarson and Yorke-Smith (2023), whereas in the next section, we show how to create a structured ensemble of INNs.

3.1. Binarized Neural Networks

The architecture of a BNN is defined by a set of layers $\mathcal{N} = \{N_0, N_1, \dots, N_L\}$, where $N_l = \{1, \dots, n_l\}$, and n_l is the number of neurons in the lth layer. Let the training set be $\mathcal{X} := \{(\boldsymbol{x}^1, y^1), \dots, (\boldsymbol{x}^t, y^t)\}$ such that $\boldsymbol{x}^i \in \mathbb{R}^{n_0}$ and $y^i \in \{-1, +1\}^{n_L}$ for every $i \in T = \{1, 2, \dots, t\}$. The first layer N_0 corresponds to the size of the input data points $\boldsymbol{x}^k$. Regarding n_L, we make the following consideration. For a classification problem with $|\mathcal{I}|$ classes, we set $n_L := \lceil \log[2]|\mathcal{I}| \rceil$. For the case $|\mathcal{I}| = 2$, therefore, n_L will be equal to one. This is consistent with binary classification problems as the two classes can be represented as +1 and −1. When $|\mathcal{I}| = 4$, then n_L will be equal to two, and the four classes will be represented by $(+1, +1), (+1, -1), (-1, +1)$, and $(-1, -1)$. When $|\mathcal{I}|$ is a power of two, the procedure generalizes in an obvious manner. However, when the number of classes is not a power of two, we still choose n_L as the nearest integer greater than or equal to the base-2 logarithm of that number, with the caveat that a single network may opt not to classify. For example, if $|\mathcal{I}| = 3$, then $n_L = 2$; $(+1, +1)$ will be associated with the first class, $(+1, -1)$ will be associated with the second class, and $(-1, +1)$ will be associated with the third class, whereas $(-1, -1)$ will be interpreted as "unclassified."

The link between neuron i in layer N_{l-1} and neuron j in layer N_l is modeled by weight $w_{ilj} \in \{-1, 0, +1\}$. Note that the binarized nature is encoded in the ± 1 weights, whereas when a weight is set to zero, the corresponding link is removed from the network. Hence, during training, we are also optimizing the architecture of the BNN.

The activation function is the binary function

$$\rho(x) := 2 \cdot \mathbb{1}(x \geq 0) - 1, \tag{1}$$

that is, a sign function reshaped such that it takes ± 1 values. Here, the indicator function $\mathbb{1}(p)$ outputs +1 if proposition p is verified and outputs 0 otherwise. This choice for the activation function has been made in line with the literature (Hubara et al. 2016).

In this paper, we aim to build different MILP models for the simultaneous training and optimization of a network architecture. To model the activation function (1) of the jth neuron of layer N_l for data point $\boldsymbol{x}^k$, we introduce a binary variable $u^k_{lj} \in \{0, 1\}$ for the indicator function $\mathbb{1}(p)$. To rescale the value of u^k_{lj} in $\{-1, +1\}$ and model the activation function value, we introduce the auxiliary variable $z^k_{lj} = (2u^k_{lj} - 1)$. For the first input layer, we set $z^k_{0j} = x^k_j$; for the last layer, we account in the loss function for whether z^k_{Lh} is different from y^k_h. The definition of the activation function becomes

$$z^k_{lj} = \rho\left(\sum_{i \in N_{l-1}} z^k_{(l-1)i} w_{ilj}\right) = 2 \cdot \mathbb{1}\left(\sum_{i \in N_{l-1}} z^k_{(l-1)i} w_{ilj} \geq 0\right) - 1 = 2u^k_{lj} - 1.$$

Notice that the activation function at layer N_l gives a nonlinear combination of the output of the neurons in the previous layer N_{l-1} and the weights w_{ilj} between the two layers. Section 4.1 shows how to formulate this activation function in terms of mixed integer linear constraints. We remark that the modeling we propose has already been presented in the literature by Toro Icarte et al. (2019).

The choice of a family of parameters $W := \{w_{ilj}\}_{l \in \{1, \dots, L\}, i \in N_{l-1}, j \in N_l}$ determines the function

$$f_W : \mathbb{R}^{n_0} \to \{\pm 1\}^{n_L}.$$

The training of a neural network is the process of computing the family W such that f_W classifies correctly both the given training data (that is, $f_W(\boldsymbol{x}^i) = y^i$ for $i = 1, \ldots, t$) and the new unlabeled testing data.

In the training of a BNN, we follow two machine learning principles for generalization: robustness and simplicity (Toro Icarte et al. 2019). In doing so, we target two objectives. (i) The resulting function f_W should generalize from the input data and be *robust* to noise in the input data. (ii) The resulting network should be *simple*: that is, with the smallest number of nonzero weights that permit us to achieve the best accuracy.

Regarding the robustness objective, there is an argument that deep neural networks have inherent robustness because minibatch stochastic gradient-based methods implicitly guide toward robust solutions (Kawaguchi et al. 2017, Keskar et al. 2017, Neyshabur et al. 2017). However, as shown in Toro Icarte et al. (2019), this is false for BNNs in a few-shot learning regime. On the contrary, MIP-based training with an appropriate objective function can generalize very well (Toro Icarte et al. 2019, Thorbjarnarson and Yorke-Smith 2023), but it does not apply to large training data sets because the size of the MIP training model is proportional to the size of the training data set.

One possible way to impose robustness in the context of few-shot learning is to maximize the margins of the neurons; that is, fixing one neuron, we aim at finding an ingoing weights configuration such that for every training input, the entry of the activation function evaluated at that neuron is confidently far away from the discontinuity point. Intuitively, neurons with larger margins require larger changes to their inputs and weights before changing their activation values. This choice is also motivated by recent work showing that margins are good predictors for the generalization of deep convolutional NNs (Jiang et al. 2019).

Regarding the simplicity objective, a significant parameter is the number of connections (Moody 1991). The training algorithm should look for an NN fitting the training data while minimizing the number of nonzero weights. This approach can be interpreted as a simultaneous compression during training, and it has been already explored in recent works (Serra et al. 2021, Roger et al. 2022).

3.1.1. MIP-Based BNN Training.

In Toro Icarte et al. (2019), two different MIP models are introduced: the Max-Margin (MM), which aims to train robust BNNs, and the Min-Weight (MW), which aims to train simple BNNs. These two models are combined with a CP model into the two hybrid methods HW and HA in order to obtain a feasible solution within a fixed time limit. Toro Icarte et al. (2019) employ CP because their MIP models do not scale as the amount of training data increases. We remark that in that work, the two objectives, robustness and simplicity, are never optimized simultaneously.

3.1.2. Gradient-Based BNN Training.

In Hubara et al. (2016), a gradient descent-based method is proposed, consisting of a local search that changes the weights to minimize a square hinge loss function. Note that a BNN trained with this approach only learns ± 1 weights. An extension of this method that exploits the same loss function but admits zero-value weights, called GD_t, is proposed in Toro Icarte et al. (2019) to facilitate the comparison with the other approaches.

3.2. Integer Neural Networks

A more general discrete NN can be obtained when the weights of the network lie in the set $\{-P, -P+1, \ldots, -1, 0, 1, \ldots, P-1, P\}$, where P is a positive integer. The resulting network is called INN, and by letting $p = 1$, we obtain the BNN presented in the previous subsection. The activation function ρ and the binary variables u_{lj}^k are defined as above. The principles leading the training are again simplicity and robustness.

An apparent advantage of using more general integer neural networks lies in the fact that the parameters have increased flexibility while still maintaining their discrete nature. Additionally, by appropriately selecting the parameter P, one can determine the number of bits used for each parameter. For instance, $p = 1$ corresponds to one bit, $p = 3$ corresponds to two bits, $p = 8$ corresponds to three bits, and in general, $P = 2^{n-1}$ corresponds to n-bits.

3.2.1. MIP-Based INN Training.

In Thorbjarnarson and Yorke-Smith (2023), three MIP models are proposed in order to train INNs. The first model, Max-Correct, is based on the idea of maximizing the number of corrected predicted images. The second model, Min-Hinge (MH), is inspired by the squared hinge loss (compare with Hubara et al. 2016). The last model, Sat-Margin (SM), combines aspects of both of the first two models. These three models always produce a feasible solution but use the margins only on the neurons of the last level, hence obtaining less robust NNs.

3.2.2. Relations to Quantized Neural Networks. By using an exact MIP solver for training INNs, we are dealing directly with the problem of training a quantized neural network, where all of the weights are restricted to take values over a small domain as discussed above. For instance, as reviewed in Gholami et al. (2022), there is a growing trend in training NNs using floating point numbers in low precision: that is, using only as few as eight bits per weight (see, for example, Banner et al. 2018). However, most of the work in the machine learning literature either focuses on the impact of low-precision arithmetic on the computation of the (stochastic) gradient and in the backpropagation algorithm or focuses on how to *quantize* a trained NN by minimizing the deterioration in the accuracy. In our work, we take a different perspective on quantization methods because we do not rely on a gradient-based method to train our INN, but we model and directly solve the problem of training the NN using only a restricted number of integer weights, which is called *integer-only quantization* in Gholami et al. (2022). Moreover, by using an exact MIP solver, we can directly find the optimal weights of our (small) INN without running the risk of being trapped into a local minima as stochastic gradient-based methods.

4. The *BeMi* Ensemble

This section first introduces a multiobjective mixed-integer linear programming (MILP) model that allows for a simultaneous training and optimization for an INN (Section 4.1), and then, it proposes a method for combining a set of neural networks for classification purposes (Section 4.2).

4.1. A Multiobjective MILP Model for Training INNs

For ease of notation, we denote with $\mathcal{L} := \{1, \dots, L\}$ the set of layers, and we denote with $\mathcal{L}_2 := \{2, \dots, L\}$, $\mathcal{L}^{L-1} := \{1, \dots, L-1\}$ two of its subsets. We also denote with $\mathfrak{b} := \max_{k \in T, j \in N_0} \{|x_j^k|\}$ a bound on the values of the training data.

4.1.1. The Multiobjective Target. A few MIP models are proposed in the literature to train INNs efficiently. In this work, to train a single INN, we use a lexicographic multiobjective function that results in the sequential solution of three different state-of-the-art MIP models: the Sat-Margin (described in Thorbjarnarson and Yorke-Smith 2023), the Max-Margin (described in Toro Icarte et al. 2019), and the Min-Weight (described in Toro Icarte et al. 2019). The first model `SM` maximizes the amount of confidently correctly predicted data. The other two models, `MM` and `MW`, aim to train an INN following two principles: robustness and simplicity. Our model is based on a lexicographic multiobjective function. First, we train an INN with the model `SM`, which is fast to solve and always gives a feasible solution. Second, we use this solution as a warm start for the `MM` model, training the INN only with the images that `SM` correctly classified. Third, we fix the margins found with `MM`, and we minimize the number of active weights with `MW`, finding the simplest INN with the robustness found by `MM`.

4.1.2. Problem Variables. The critical part of our model is the formulation of the nonlinear activation function (1). We use an integer variable $w_{ilj} \in \{-P, -P+1, \dots, P\}$ to represent the weight of the connection between neuron $i \in N_{l-1}$ and neuron $j \in N_l$. Variable u_{lj}^k models the result of the indicator function $\mathbb{1}(p)$ that appears in the activation function $\rho(\cdot)$ for the training instance $\boldsymbol{x}^k$. The neuron activation is actually defined as $2u_{lj}^k - 1$. We introduce auxiliary variables c_{ilj}^k to represent the products $c_{ilj}^k = (2u_{lj}^k - 1)w_{ilj}$. Note that although in the first layer, these variables share the same domain of the inputs, from the second layer on, they take values in $\{-P, -P+1, \dots, P\}$. Finally, the auxiliary variables $\hat{y}^k$ represent a predicted label for the input $\boldsymbol{x}^k$, and variables q_j^k are used to take into account the data points correctly classified.

The procedure is designed such that the parameter configuration obtained in the first step is used as a warm start for the `MM`. Similarly, the solution of the second step is used as a warm start for the solver to solve `MW`. In this case, the margins lose their nature as decision variables and become deterministic constants derived from the solution of the previous step.

4.1.3. Sat-Margin Model. We first train our INN using the following `SM` model:

$$\max \quad \sum_{k \in T} \sum_{j \in N_L} q_j^k \tag{2a}$$

$$\text{s.t.} \quad q_j^k = 1 \Rightarrow \hat{y}_j^k \cdot y_j^k \geq \frac{1}{2} \qquad \forall j \in N_L, k \in \mathrm{T}, \tag{2b}$$

$$q_j^k = 0 \Rightarrow \hat{y}_j^k \cdot y_j^k \leq \frac{1}{2} - \hat{\epsilon} \qquad \forall j \in N_L, k \in T, \tag{2c}$$

$$\hat{y}_j^k = \frac{2}{P \cdot (n_{L-1}+1)} \sum_{i \in N_{L-1}} c_{iLj}^k \qquad \forall j \in N_L, k \in T, \tag{2d}$$

$$u_{lj}^k = 1 \Rightarrow \sum_{i \in N_{l-1}} c_{ilj}^k \geq 0 \qquad \forall l \in \mathcal{L}^{L-1}, j \in N_l, k \in T, \tag{2e}$$

$$u_{lj}^k = 0 \Rightarrow \sum_{i \in N_{l-1}} c_{ilj}^k \leq -\epsilon \qquad \forall l \in \mathcal{L}^{L-1}, j \in N_l, k \in T, \tag{2f}$$

$$c_{i1j}^k = x_i^k \cdot w_{i1j} \qquad \forall i \in N_0, j \in N_1, k \in T, \tag{2g}$$

$$c_{ilj}^k = (2u_{(l-1)j}^k - 1) w_{ilj} \qquad \forall l \in \mathcal{L}_2, i \in N_{l-1}, j \in N_l, k \in T, \tag{2h}$$

$$q_j^k \in \{0, 1\} \qquad \forall j \in N_L, k \in T, \tag{2i}$$

$$w_{ilj} \in \{-P, -P+1, \ldots, P\} \qquad \forall l \in \mathcal{L}, i \in N_{l-1}, j \in N_l, \tag{2j}$$

$$u_{lj}^k \in \{0, 1\} \qquad \forall l \in \mathcal{L}^{L-1}, j \in N_l, k \in T, \tag{2k}$$

$$c_{i1j}^k \in [-P \cdot \mathfrak{b}, P \cdot \mathfrak{b}] \qquad \forall i \in N_0, j \in N_1, k \in T, \tag{2l}$$

$$c_{ilj}^k \in \{-P, -P+1, \ldots, P\} \qquad \forall l \in \mathcal{L}_2, i \in N_{l-1}, j \in N_l, k \in T, \tag{2m}$$

with $\hat{\epsilon} := \frac{\epsilon}{2P \cdot (n_{L-1}+1)}$. The objective function (2a) maximizes the number of data points that are correctly classified. Note that ϵ is a small quantity standardly used to model strict inequalities. The implication Constraints (2b) and (2c) and Constraints (2d) are used to link the output $\hat{y}_j^k$ with the corresponding variable q_j^k appearing in the objective function. The implication Constraints (2e) and (2f) model the result of the indicator function for the kth input data. Constraints (2g) and the bilinear Constraints (2h) propagate the results of the activation functions within the neural network. We linearize all these constraints with standard big-M techniques (Williams 2013).

The solution of Model (2a)–(2m) gives us the solution vectors $\mathbf{c}_{\mathtt{SM}}$, $\mathbf{u}_{\mathtt{SM}}$, $\mathbf{w}_{\mathtt{SM}}$, $\hat{\mathbf{y}}_{\mathtt{SM}}$, $\mathbf{q}_{\mathtt{SM}}$. We then define the set

$$\hat{T} = \{k \in T \mid q_{j\,\mathtt{SM}}^k = 1, \ \forall j \in N_L\} \tag{3}$$

of confidently correctly predicted images. We use these images as input for the next Max-Margin, and we use the vector of variables $\mathbf{c}_{\mathtt{SM}}$, $\mathbf{u}_{\mathtt{SM}}$, $\mathbf{w}_{\mathtt{SM}}$ to warm start the solution of MM.

4.1.4. Max-Margin Model. The second level of our lexicographic multiobjective model maximizes the overall margins of every single neuron activation, with the ultimate goal of training a robust INN. Starting from the model SM, we introduce the margin variables m_{lj}, and we introduce the following Max-Margin model:

$$\max \sum_{l \in \mathcal{L}} \sum_{j \in N_l} m_{lj} \tag{4a}$$

$$\text{s.t. (2g)–(2m)} \qquad \forall k \in \hat{T},$$

$$\sum_{i \in N_{L-1}} y_j^k c_{iLj}^k \geq m_{Lj} \qquad \forall j \in N_L, k \in \hat{T}, \tag{4b}$$

$$u_{lj}^k = 1 \Rightarrow \sum_{i \in N_{l-1}} c_{ilj}^k \geq m_{lj} \qquad \forall l \in \mathcal{L}^{L-1}, j \in N_l, k \in \hat{T}, \tag{4c}$$

$$u_{lj}^k = 0 \Rightarrow \sum_{i \in N_{l-1}} c_{ilj}^k \leq -m_{lj} \qquad \forall l \in \mathcal{L}^{L-1}, j \in N_l, k \in \hat{T}, \tag{4d}$$

$$m_{lj} \geq \epsilon \qquad \forall l \in \mathcal{L}, j \in N_l. \tag{4e}$$

Again, we can linearize Constraints (4c) and (4d) with standard big-M constraints. This model gives us the solution vectors $\mathbf{c}_{\texttt{MM}}, \mathbf{u}_{\texttt{MM}}, \mathbf{w}_{\texttt{MM}}, \mathbf{m}_{\texttt{MM}}$. We then evaluate $\mathbf{v}_{\texttt{MM}}$ as

$$v_{ilj\texttt{MM}} = \begin{cases} 0 & \text{when } w_{ilj\texttt{MM}} = 0, \\ 1 & \text{otherwise,} \end{cases} \qquad \forall l \in \mathcal{L}, i \in N_{l-1}, j \in N_l. \tag{5}$$

4.1.5. Min-Weight Model. The third level of our multiobjective function minimizes the overall number of nonzero weights: that is, the connections of the trained INN. We introduce the new auxiliary binary variable v_{ilj} to model the presence or absence of the link w_{ilj}. Starting from the solution of model MM, we fix $\hat{\mathbf{m}} = \mathbf{m}_{\texttt{MM}}$, and we pass the solution $\mathbf{c}_{\texttt{MM}}, \mathbf{u}_{\texttt{MM}}, \mathbf{w}_{\texttt{MM}}, \mathbf{v}_{\texttt{MM}}$ as a warm start to the following MW model:

$$\min \quad \sum_{l \in \mathcal{L}} \sum_{i \in N_{l-1}} \sum_{j \in N_l} v_{ilj} \tag{6a}$$

$$\text{s.t.} \quad \text{(2g)–(2m)} \qquad \forall k \in \hat{T},$$

$$\sum_{i \in N_{L-1}} y_j^k c_{iLj}^k \geq \hat{m}_{Lj} \qquad \forall j \in N_L, k \in \hat{T}, \tag{6b}$$

$$u_{lj}^k = 1 \Rightarrow \sum_{i \in N_{l-1}} c_{ilj}^k \geq \hat{m}_{lj} \qquad \forall l \in \mathcal{L}^{L-1}, j \in N_l, k \in \hat{T}, \tag{6c}$$

$$u_{lj}^k = 0 \Rightarrow \sum_{i \in N_{l-1}} c_{ilj}^k \leq -\hat{m}_{lj} \qquad \forall l \in \mathcal{L}^{L-1}, j \in N_l, k \in \hat{T}, \tag{6d}$$

$$-v_{ilj} \cdot P \leq w_{ilj} \leq v_{ilj} \cdot P \qquad \forall l \in \mathcal{L}, i \in N_{l-1}, j \in N_l, \tag{6e}$$

$$v_{ilj} \in \{0, 1\} \qquad \forall l \in \mathcal{L}, i \in N_{l-1}, j \in N_l. \tag{6f}$$

Note that whenever v_{ilj} is equal to zero, the corresponding weight w_{ilj} is set to zero because of Constraint (6e), and hence, the corresponding link can be removed from the network.

4.1.6. Lexicographic Multiobjective. By solving the three models SM, MM, and MW sequentially, we first maximize the amount of input data that is correctly classified; then, we maximize the margin of every activation function, and finally, we minimize the number of nonzero weights. The solution of the decision variables w_{ilj} of the last model MW defines our classification function $f_W : \mathbb{R}^{n_0} \to \{\pm 1\}^{n_L}$.

4.2. The *BeMi* Structure

Having explained the various MIP models of INNs, we next introduce our ensemble approach for MIP-based training of INNs.

4.2.1. Ensemble. Define $\mathcal{P} := \{\{i, j\} \text{ s.t. } i \neq j,\ i, j \in \mathcal{I}\}$ as the set of all of the subsets of the set $\mathcal{I}$ that have cardinality 2, where $\mathcal{I}$ is the set of the classes of the classification problem. Then, our structured ensemble is constructed in the following way.

1. We train an INN denoted by $\mathcal{N}_{ij}$ for every $\{i, j\} \in \mathcal{P}$ (i.e., for each possible pair of elements of $\mathcal{I}$).
2. When testing a data point, we feed it to our list of trained INNs, obtaining a list of predicted labels; namely, we obtain the predicted label $\mathfrak{e}_{ij}$ from the network $\mathcal{N}_{ij}$.
3. We then apply a majority voting system.

The idea behind this structured ensemble is that given an input $\boldsymbol{x}^k$ labeled l $(= y^k)$, the input is fed into $\binom{n}{2}$ networks where $n-1$ of them are trained to recognize an input with label l. If all of the networks correctly classify the input $\boldsymbol{x}^k$ as l, then at most, $n-2$ other networks can classify the input with a different label $l' \neq l$, and so, the input is correctly labeled with the most occurring label l. With this approach, if we plan to use $r \in \mathbb{N}$ inputs for each label, we are feeding each of our INNs a total of $2 \cdot r$ inputs instead of feeding $n \cdot r$ inputs to a single large INN. Clearly, when training the networks $\mathcal{N}_{ij}$ and the network $\mathcal{N}_{ik}$, the inputs of the class i are the same, so we only need a total of r inputs for each class. When $n \gg 2$, it is much easier to train our structured ensemble of INNs than train one large INN because of the fact that the MILP model size depends linearly on the amount of input data.

4.2.2. Majority Voting System. After the training, we feed one input x^k to our list of INNs, and we need to elaborate on the set of outputs.

Definition 1 (Dominant Label). For every $b \in \mathcal{I}$, we define

$$C_b = \{\{i,j\} \in \mathcal{P} \mid \mathfrak{e}_{ij} = b\},$$

and we say that a label b is a *dominant label* if $|C_b| \geq |C_l|$ for every $l \in \mathcal{I}$. We then define the set of dominant labels

$$\mathcal{D} := \{b \in \mathcal{I} \mid b \text{ is a dominant label}\}.$$

Using this definition, we can have three possible outcomes.

a. There exists a label $i \in \mathcal{I}$ such that $\mathcal{D} = \{i\} \Rightarrow$ our input is labeled as i.
b. There exist $j,k \in \mathcal{I}$ such that $\mathcal{D} = \{j,k\} \Rightarrow$ our input is labeled as $\mathfrak{e}_{jk}$.
c. There exist more than two dominant labels $\Rightarrow$ our input is not classified.

Although case (a) is straightforward, we can label our input even when we do not have a clear winner: that is, when we have trained an INN on the set of labels that are the most frequent (i.e., case (b)). Note that the proposed structured ensemble alongside its voting scheme can also be exploited for regular NNs.

Definition 2 (Label Statuses). In our labeling system, when testing an input, seven different cases (herein called *label statuses*) can arise. The statuses names are of the form "number of the dominant labels + fairness of the prediction." The first parameter can be one, two, or o, where o means "other cases." The fairness of the prediction is C when it is correct or I when it is incorrect. The superscripts related to I' and I'' only distinguish between the different cases. These cases are described through the tree diagram in Figure 1.

The cases in which the classification algorithm classifies correctly are, therefore, only (1C) and (2C). Note that every input test will fall into exactly one label status.

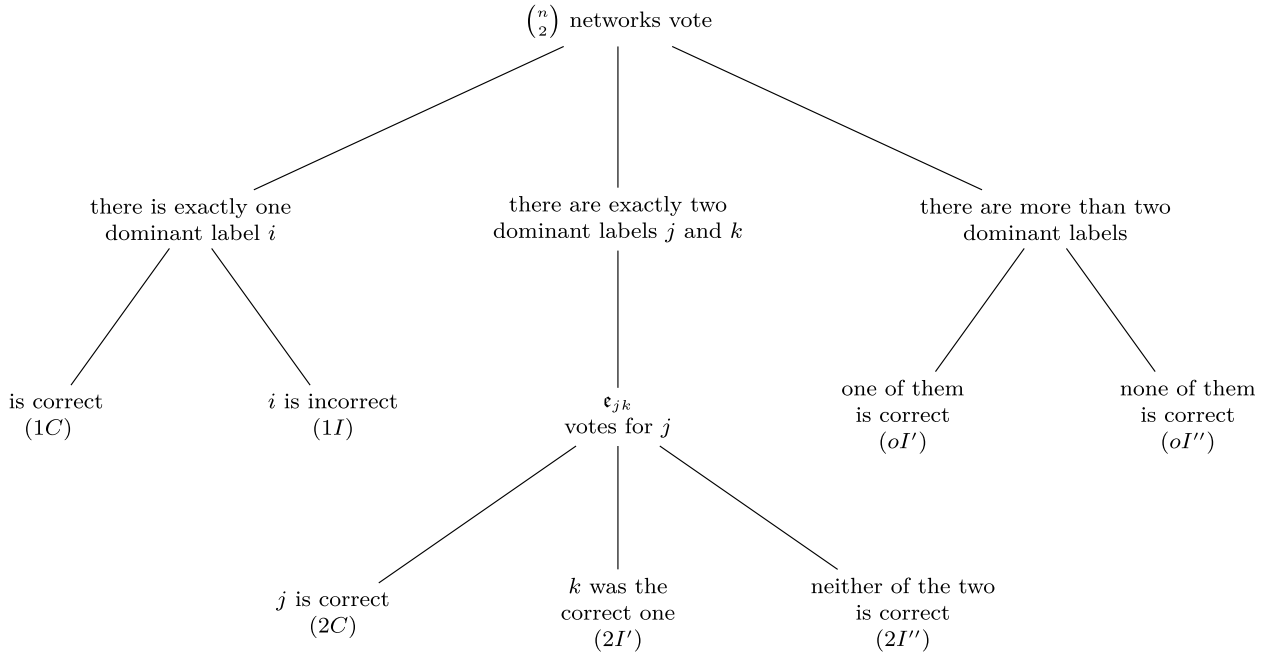
Example 1. Let us take $\mathcal{I} = \{bird, cat, dog, frog\}$. Note that in this case, we have to train $\binom{4}{2} = 6$ networks:

$$\mathcal{N}_{\{bird,\,cat\}},\ \mathcal{N}_{\{bird,\,dog\}},\ \mathcal{N}_{\{bird,\,frog\}},\ \mathcal{N}_{\{cat,\,dog\}},\ \mathcal{N}_{\{cat,\,frog\}},\ \mathcal{N}_{\{dog,\,frog\}},$$

where the first one distinguishes between *bird* and *cat*, the second one distinguishes between *bird* and *dog*, and so on. A first input could have the following predicted labels:

$$\mathfrak{e}_{\{bird,\,cat\}} = bird, \quad \mathfrak{e}_{\{bird,\,dog\}} = bird, \quad \mathfrak{e}_{\{bird,\,frog\}} = frog,$$
$$\mathfrak{e}_{\{cat,\,dog\}} = cat, \quad \mathfrak{e}_{\{cat,\,frog\}} = cat, \quad \mathfrak{e}_{\{dog,\,frog\}} = dog.$$

Figure 1. Tree Diagram of Label Statuses



We would then have

$$C_{bird} = \{\{bird,\ cat\}, \{bird,\ dog\}\}, \qquad C_{cat} = \{\{cat,\ dog\}, \{cat,\ frog\}\},$$
$$C_{dog} = \{\{dog,\ frog\}\}, \qquad C_{frog} = \{\{bird,\ frog\}\}.$$

In this case, $\mathcal{D} = \{bird, cat\}$ because $|C_{bird}| = |C_{cat}| = 2 > 1 = |C_{dog}| = |C_{frog}|$, and we do not have a clear winner; however, because $|\mathcal{D}| = 2$, we have trained a network that distinguishes between the two most voted labels, and so, we use its output as our final predicted label, labeling our input as $\mathfrak{e}_{\{bird,\ cat\}} = bird$. If *bird* is the right label, we are in label status (2*C*); if the correct label is *cat*, we are in label status (2*I*′). Otherwise, we are in label status (2*I*″).

Example 2. Let us take $\mathcal{I} = \{0, 1, \ldots, 9\}$. Note that in this case, we have to train $\binom{10}{2} = 45$ networks and that $|C_b| \leq 9$ for all $b \in \mathcal{I}$. Hence, an input could be labeled as follows:

$$C_0 = (\{0, i\})_{i=1,2,3,5,7,8}, \qquad C_1 = (\{1, i\})_{i=5,6}, \qquad C_2 = (\{2, i\})_{i=1,5,8},$$
$$C_3 = (\{3, i\})_{i=1,2,4,5}, \qquad C_4 = (\{4, i\})_{i=0,1,2,5,6,7,9}, \qquad C_5 = (\{5, i\})_{i=6,7},$$
$$C_6 = (\{6, i\})_{i=0,2,3,7}, \qquad C_7 = (\{7, i\})_{i=1,2,3},$$
$$C_8 = (\{8, i\})_{i=1,3,4,5,6,7}, \qquad C_9 = (\{9, i\})_{i=0,1,2,3,5,6,7,8}.$$

Visually, we can represent an input being labeled as above with the scheme in Figure 2. In Figure 2, we have omitted the name of each element of the set C_i for simplicity; for example, the dots above C_1 in Figure 2 represent the sets $\{1, 5\}, \{1, 6\}$. Because $\mathcal{D} = \{9\}$, our input is labeled as nine. If nine is the right label, we are in label status (1*C*); if nine is the wrong label, we are in label status (1*I*). If instead, $\hat{C}_j = C_j$, $j = 0, \ldots, 7$, and

$$\hat{C}_8 = (\{8, i\})_{i=1,3,4,5,6,7,9}, \quad \hat{C}_9 = (\{9, i\})_{i=0,1,2,3,5,6,7},$$

then $|\hat{\mathcal{D}}| = |\{4, 8, 9\}| = 3$ so that our input was labeled as -1. If the correct label is four, eight, or nine, we are in label status (*mI*′); otherwise, we are in label status (*mI*″). Lastly, if $\overline{C}_j = C_j$, $j \in \{0, 1, 2, 4, 5, 6, 7, 8\}$, and

$$\overline{C}_3 = (\{3, i\})_{i=1,2,4,5,9}, \quad \overline{C}_9 = (\{9, i\})_{i=0,1,2,5,6,7,8},$$

then $|\overline{\mathcal{D}}| = |\{4, 9\}| = 2$, and because $\{4, 9\} \in \overline{C}_4$, our input is labeled as four. If four is the correct label, we are in label status (2*C*). If nine is the correct label, we are in label status (2*I*′). Otherwise, we are in label status (2*I*″). Note that for brevity, in this example we used the notation $(\{j, i\})_{i=i_1,\ldots,i_n} = \{\{j, i_1\}, \ldots, \{j, i_n\}\}$, $j, i_1, \ldots, i_n \in \{0, \ldots, 9\}$.

5. Empirical Study

Having introduced the *BeMi* approach, we now undertake a series of six experiments in order to explore the following questions.

Experiment 1. What is the impact of a threefold multiobjective model compared with a twofold or single-objective model? (Recall Section 4.)

Experiment 2. How does the *BeMi* ensemble compare with the previous state-of-the-art MIP models for training BNNs in the context of few-shot learning?

Experiment 3. How does the *BeMi* ensemble scale with the number of training images, considering two different types of BNNs?

Experiment 4. How does the *BeMi* ensemble perform on various data sets, comparing the running time, the average gap to the optimal training MILP model, and the percentage of links removed?

Figure 2. (Color online) Example of Voting Scheme

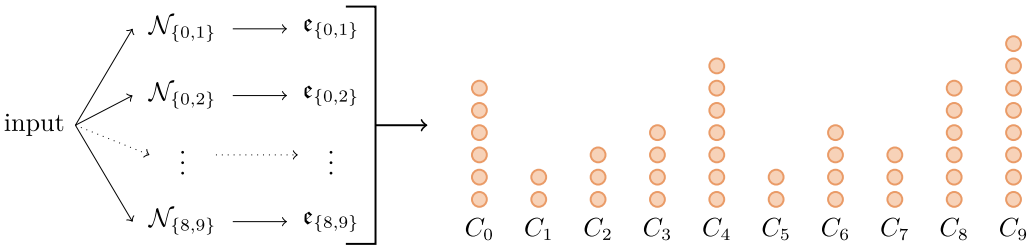


Table 1. Details of the Different Data Sets Exploited in the Experiments

Data set	Number of classes	Input dimension	Data values	No. of training set	No. of test set
MNIST	10	28×28	Integers	60,000	10,000
Fashion-MNIST	10	28×28	Integers	60,000	10,000
Heart disease	2	13	Continuous	$920 - x$	x

Experiment 5. What are the performance differences between a nontrivial INN and a BNN? Do INNs exhibit particular weight distribution characteristics? A state-of-the-art comparison is also provided.

Experiment 6. How does the *BeMi* ensemble perform on a continuous, low-dimensional data set comparing BNNs and nontrivial INNs? Do INNs exhibit the same weight distribution characteristics found in Experiment 5?

5.1. Data Sets

Three data sets are adopted for the experiments. We use first the standard MNIST data set (LeCun et al. 1998) for a fair comparison with the literature, and second, we use the larger Fashion-MNIST data set (Xiao et al. 2017). For these two MNIST data sets, we test our results on 800 images for each class in order to have the same amount of test data for every class. Note that the MNIST data set has 10,000 test data, but they are not uniformly distributed over the 10 classes. For each experiment, we report the average over five different samples of images (i.e., we perform five different trainings, and we report the average over them while testing the same images). The images are sampled uniformly at random in order to avoid overlapping between different experiments. Beyond MNIST, we use the Heart Disease data set (Janosi et al. 1988) from the University of California, Irvine (UCI) repository. Table 1 summarizes the data sets.

5.2. Implementation Details

As the solver, we use Gurobi version 10.0.1 (Gurobi Optimization LLC 2023) to solve our MILP models. The solver parameters are left to the default values if not specified otherwise. Apart from the first experiment, where we chose to consider every model equally, the fraction of time given to each step of the multiobjective model has been chosen accordingly to the importance of finding a feasible and robust solution. All of the MILP experiments were run on an high-performance computing cluster running CentOS using a single node per experiment. Each node has an Intel CPU with eight physical cores working at 2.1 GHz and 16 GB of RAM. In all of our experiments concerning integer-value data sets, we fix the value $\epsilon = 0.1$. Notice that because of the integer nature of the weights, the image of the activation function, and other data, setting ϵ equal to any number smaller than one is equivalent. When using continuous-value data sets, we fix the value $\epsilon = 1 \cdot 10^{-6}$ in accordance with the default variable precision tolerance of the Gurobi MILP solver we will use. The source code is available on GitHub (Bernardelli et al. 2024).

5.3. Time Limit Management

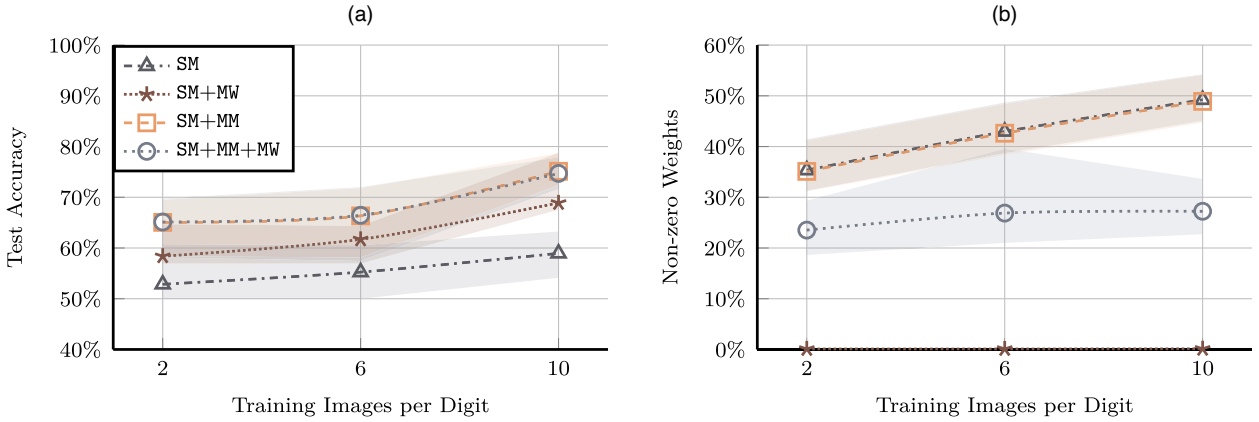
Concerning the time limits for the different optimization models, the following choices have been made. In Experiments 1 and 6, the time limit is equally distributed between the three models to have a fair comparison. In Experiments 2, 3, 4, and 5, the majority of the imposed time limit was given to the first two models. The first model ensures the feasibility of the whole pipeline and maximizes the number of correctly classified images in the training phase, and it was considered important in the context of few-shot learning because we do not have lots of images as training inputs. The second model was given a bigger time limit too because preliminary results, also shown in previous works, highlight the fact that the maximization of the margin ensures a better test accuracy with respect to the minimization of the links. In addition to this, the overall time limits have been chosen based on two criteria. Where comparisons with the literature are made, the selection ensures a fair comparison. In the remaining cases, the choice of time limit has been empirical, aiming to highlight the algorithm's quality.

5.4. Experiment 1

The goal of the first experiment is to study the impact of the multiobjective model composed of `SM`, `MM`, and `MW` with respect to the models composed of `SM` and `MM`, the one composed of `SM` and `MW`, and the one composed of only `SM`.

The results refer to a BNN specialized in distinguishing between digits 4 and 9 of the MNIST data set. These two digits were chosen because their written form can be quite similar. Indeed, among all 10 digits, digits 4 and 9 are very often mistaken for each other as is shown in the confusion matrix in Appendix B.

Figure 3. (Color online) The SM + MM + MW Model Achieves the Same Accuracy as the SM + MM Model, Outperforming the SM Model, While Having the Smallest Percentage of Nonzero Weights Apart from the SM + MW Model, Which Has an Almost-Zero Percentage of Nonzero Weights but also Has a Lower Accuracy of the Models That Maximize the Margins



Notes. The dotted lines represent the average accuracy obtained over five instances, whereas the shaded areas highlight the range between the minimum and maximum values. (a) Test accuracy of different models. (b) Simplicity of different models.

The NN architecture consists of two hidden layers and has $[784, 4, 4, 1]$ neurons. The architecture is chosen to mimic the one used in Toro Icarte et al. (2019): that is, $[784, 16, 16, 10]$ but with fewer neurons. The number of training images varies between 2, 6, and 10, whereas the test images are 800 per digit and therefore, 1,600 in total. The imposed time limit is 30 minutes, and it is equally distributed in the steps of each model: 30 minutes for the SM model, 15 minutes + 15 minutes for the SM + MW model, 15 minutes + 15 minutes for the SM + MM model, and 10 minutes + 10 minutes + 10 minutes for the SM + MM + MW model.

Figure 3(a) compares the test accuracy of the four hierarchical models, showing how the MM model ensures an increase in accuracy, whereas the MW allows the network to be pruned without performance being affected. Figure 3(b) displays the percentages of nonzero weights of the three trained models. Note that in this case, the training accuracy is always 100%, and so, we did not add it to the plot. Also, the dotted lines in Figure 3 represent the average accuracy obtained over five instances, whereas the shaded areas in Figure 3 highlight the range between the minimum and maximum values. This will be the case for every other plot if not specified otherwise. The MW step allows the number of nonzero weights to drop without accuracy being affected, hence resulting in an effective pruning. This behavior is also observed for other couples of digits, even the ones that are easier to distinguish, namely one and eight. Results of this experiment are reported in Appendix B.

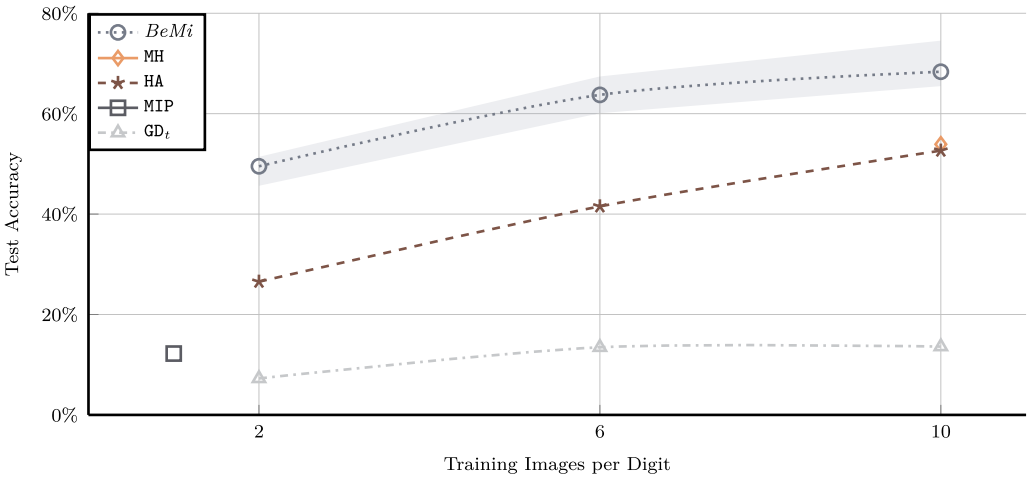
Based on these reasons, for the remaining experiments, we will exclusively employ the model that incorporates all three steps.

5.5. Experiment 2

The goal of the second experiment is to compare the *BeMi* ensemble with the following state-of-the-art methods: the pure MIP model in Toro Icarte et al. (2019), the hybrid CP-MIP model based on Max-Margin optimization (HA) in Toro Icarte et al. (2019), the gradient-based method GD_t introduced in Hubara et al. (2016) and adapted in Toro Icarte et al. (2019) to deal with link removal, and the Min-Hinge model proposed in Thorbjarnarson and Yorke-Smith (2023). For the comparison, we fix the setting of Toro Icarte et al. (2019), which takes from the MNIST up to 10 images for each class for a total of 100 training data points and which uses a time limit of 7,200 seconds to solve their MIP training models.

In our experiments, we train the *BeMi* ensemble with 2, 6, and 10 samples for each digit. Because our ensemble has 45 BNNs, we leave for the training of each single BNN a maximum of 160 seconds (because $160 \cdot 45 = 7,200$). In particular, we give a 75-second time limit to the solution of SM, a 75-second time limit to MM, and a 10-second time limit to MW. In all of our experiments, whenever the optimum is reached within the time limit, the remaining time is added to the time limit of the subsequent model. We remark that our networks could be trained in parallel, which would highly reduce the wall-clock run time. For the sake of completeness, we note that we are using $45 \cdot (784 \cdot 4 + 4 \cdot 4 + 4 \cdot 1) = 142,020$ parameters (all of the weights of all the 45 BNNs) instead of the $784 \cdot 16 + 16 \cdot 16 + 16 \cdot 10 = 12,960$ parameters used in Toro Icarte et al. (2019) for a single large BNN. Note that in this case, the dimension of the parameter space is $3^{12,960} (\cong 10^{6,183})$, whereas in our case, it is $45 \cdot 3^{3,156} (\cong 10^{1,507})$. In the first case, the solver has

Figure 4. (Color online) Comparison of Published Approaches vs. *BeMi* in Terms of Accuracy over the MNIST Data Set Using Few-Shot Learning with 2, 6, and 10 Images per Digit



to find an optimal solution between all of the $10^{6,183}$ different parameter configurations, whereas with the *BeMi* ensemble, the solver has to find 45 optimal solutions, each of which lives in a set of cardinality $10^{1,507}$. This significantly improves the solver performances. We remark that a parameter configuration is given by a weight assignment $\hat{W} = (\hat{w}_{ijl})_{ijl}$ because every other variable is uniquely determined by $\hat{W}$.

Figure 4 compares the results of our *BeMi* ensemble with the four other methods presented above. Note that the pure MIP model in Toro Icarte et al. (2019) can handle a single image per class in the given time limit, and so, only one point is reported, and note also that for the minimum hinge model `MH` presented in Thorbjarnarson and Yorke-Smith (2023), only the experiment with 10 digits per class was performed. We report the best results reported in the original papers for these four methods.

The *BeMi* ensemble obtains an average accuracy of 68%, and it outperforms all other approaches when 2, 6, or 10 digits per class are used. Note that our method attains 100% accuracy on the training set; that is, the `SM` model correctly classifies all the images. In this case, the first model is not needed to ensure feasibility, but it serves mainly as a warm start for the `MM` model.

5.6. Experiment 3

The goal of the third experiment is to study how our approach scales with the number of data points (i.e., images) per class and how it is affected by the architecture of the small BNNs within the *BeMi* ensemble. For the number of data points per class, we use 10, 20, 30, 40 training images per digit. We use the layers $\mathcal{N}_a = [784, 4, 4, 1]$ and $\mathcal{N}_b = [784, 10, 3, 1]$ for the two architectures. Although the first architecture is chosen to be consistent with the previous experiments, the second one can be described as an integer approximation of $[N, \log_2 N, \log_2(\log_2 N), \log_2(\log_2(\log_2 N))]$. Herein, we refer to Experiments 3a and 3b as the two subsets of experiments related to the architectures $\mathcal{N}_a$ and $\mathcal{N}_b$. In both cases, we train each of our 45 BNNs with a time limit of 290 seconds for model `SM`, 290 seconds for model `MM`, and 20 seconds for model `MW` for a total of 600 seconds (i.e., 10 minutes for each BNN).

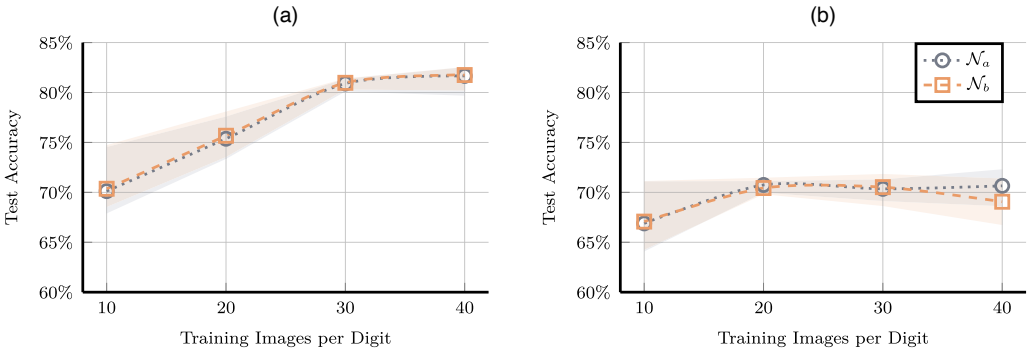
Figure 5(a) shows the results for Experiments 3a and 3b. The dotted line and the dashed line in Figure 5(a) refer to the two average accuracies of the two architectures, whereas the colored areas in Figure 5(a) include all the accuracy values obtained as the training instances vary. The two architectures behave similarly, and the best average accuracy exceeds 81%.

Table 2 reports the results for the *BeMi* ensemble, where we distinguish among images classified as correct, wrong, or unclassified. These three conditions refer to different label statuses specified in Definition 2. The correct labels are the sums of the statuses (1*C*) and (2*C*); the wrong labels are the sums of the statuses (2*I*′), (2*I*″), and (1*I*); and the unclassified labels (*n.l.*) are the sums of the statuses (*oI*′) and (*oI*″). Notice that the vast majority of the test images have only one dominant label and so, fall into statuses (1*C*) or (1*I*). The unclassified images are less than 2.31%.

5.7. Experiment 4

The goal of the fourth experiment is to revisit the questions of Experiments 3a and 3b with the two architectures $\mathcal{N}_a$ and $\mathcal{N}_b$ using the more challenging Fashion-MNIST data set.

Figure 5. (Color online) Average Accuracy for the *BeMi* Ensemble Tested on Two Architectures, Namely $\mathcal{N}_a = [784, 4, 4, 1]$ and $\mathcal{N}_b = [784, 10, 3, 1]$, Using 10, 20, 30, 40 Images per Class



Notes. (a) MNIST. (b) Fashion-MNIST.

Figure 5(b) shows the results of Experiments 3a and 3b. As in Figure 4, the dotted line and the dashed line in Figure 5(b) represent the average percentages of correctly classified images, whereas the colored areas in Figure 5(b) include all accuracy values obtained as the instances vary. For the Fashion-MNIST, the best average accuracy exceeds 70%.

Table 3 reports detailed results for all Experiments 2 and 3. The first two columns in Table 3 give the data set and the architecture, the third column in Table 3 reports the total number of links (i.e., the total number of w variables) for each architecture, and the fourth column in Table 3 specifies the number of images per digit used during training. The fifth column in Table 3 reports the run time for solving model `SM`. Note that the time limit is 290 seconds; hence, we solve exactly the first model, consistently achieving a training accuracy of 100%. The remaining five columns are as follows. Gap (percentage) refers to the mean and maximum percentage gaps at the second MILP model (`MM`) of our lexicographic multiobjective model as reported by the Gurobi `MIPgap` attribute. The column links (percentage) indicates the percentage of nonzero weights after the solution of the second model `MM` and after the solution of the last model `MW`. The column active links indicates the total number of nonzero weights after the solution of the last model `MW`. The results show that the run time and the gap increase with the size of the input set. However, for the percentage of removed links, there is a significant difference between the two data sets; for MNIST, our third model `MW` removes around 70% of the links, whereas for the Fashion-MNIST, it removes around 50% of the links. Note that in both cases, these significant reductions show how our model is also optimizing the BNN architecture. Furthermore, note that even if the accuracy of the two architectures is comparable, the total number of nonzero weights of $\mathcal{N}_a$ is about half the number of nonzero weights of $\mathcal{N}_b$.

5.8. Experiment 5

The goal of the fifth experiment is to compare the performances of BNNs and nontrivial INNs. The results refer to five different runs of an INN of architecture $[784, 4, 4, 1]$ specialized in distinguishing between digits 4 and 9 of the MNIST data set. The number of training images varies between 2, 6, 10, 20, 30, and 40, whereas the test images are 800 per digit and therefore, 1,600 in total. The imposed time limit is 290 seconds + 290 seconds + 20 seconds.

As Figure 6 shows, different INNs are comparable not only in the average accuracy, represented by the dotted line, but also in the maximum and minimum accuracy as reported by the shaded areas.

Table 2. Percentages of MNIST Images Classified as Correct, Wrong, or Unclassified and Percentages of Label Statuses for the Architecture $\mathcal{N}_a = [784, 4, 4, 1]$

	Classification (%)			Label status (%)						
Images per class	Correct	Wrong	*n.l.*	(1C)	(1*I*)	(2C)	(2*I*′)	(2*I*″)	(*oI*′)	(*oI*″)
10	70.12	27.65	2.23	68.43	24.53	1.69	1.21	1.91	1.88	0.35
20	75.37	22.32	2.31	73.79	19.33	1.58	1.39	1.60	2.02	0.29
30	80.90	17.46	1.64	79.64	15.01	1.26	1.25	1.20	1.46	0.18
40	81.66	16.68	1.66	80.34	14.36	1.32	1.09	1.23	1.45	0.21

Notes. The vast majority of the test images have only one dominant label, and so, they fall into status (1C) or (1*I*). The unclassified images (*n.l.*) are less than 2.31%.

Table 3. Aggregate Results for Experiments 2 and 3

Data set	Layers	Total links	Images per class	Model SM time (seconds)	Gap (%)		Links (%)		Active links
					Mean	Max	(MM)	(MW)	
MNIST	784, 4, 4, 1	3,156	10	3.00	12.06	20.70	49.13	29.21	921.80
			20	6.47	14.81	22.18	54.70	28.41	896.60
			30	10.60	16.04	24.08	56.44	30.46	961.40
			40	15.04	15.98	26.22	57.92	29.27	923.80
	784, 10, 3, 1	7,873	10	6.04	4.52	7.37	49.28	24.72	1,946.20
			20	15.01	5.46	8.40	54.97	24.40	1,921.00
			30	22.68	5.92	11.00	56.73	26.96	2,122.60
			40	33.97	6.21	20.87	58.29	24.66	1,941.40
Fashion-MNIST	784, 4, 4, 1	3,156	10	4.83	13.34	26.48	87.75	58.76	1,854.40
			20	9.77	14.05	28.91	90.73	59.97	1,892.60
			30	36.10	19.95	136.33	92.41	58.12	1,834.20
			40	72.15	30.36	333.70	93.57	59.46	1,876.60
	784, 10, 3, 1	7,873	10	11.42	4.87	9.14	87.90	51.57	4,060.20
			20	21.46	5.11	9.86	91.05	52.29	4,116.80
			30	35.12	6.30	40.07	92.78	52.62	4,142.80
			40	52.37	8.99	56.14	94.01	53.38	4,202.60

Notes. The fifth column reports the run time to solve the first model SM. Gap (percentage) refers to the mean and maximum percentage gaps at the second MILP model MM. The column links (percentage) indicates the percentage of nonzero weights after the solution of models MM and MW. The column active links indicates the total number of nonzero weights after the solution of model MW, as always averaged over five instances.

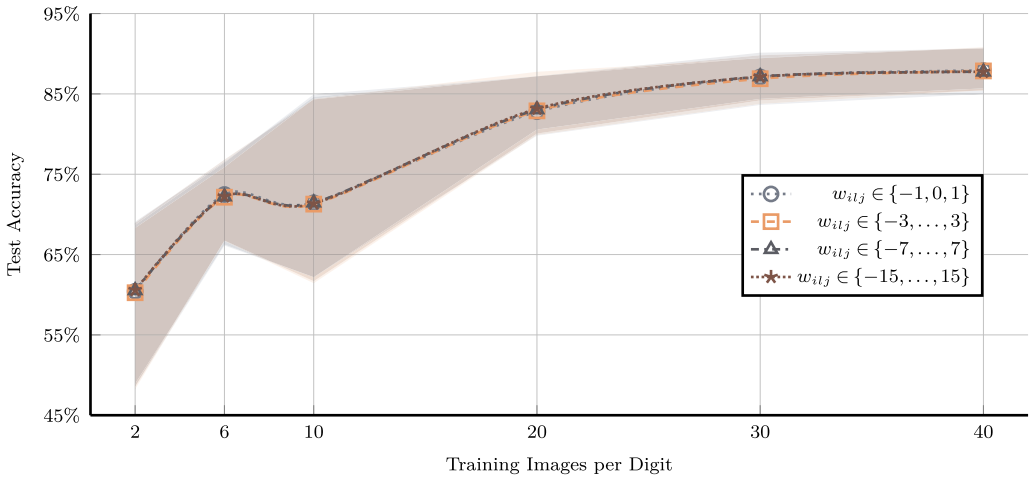
In order to study why different values of P lead to comparable accuracy, we report the weight distributions of the INNs whose accuracy is depicted in Figure 6. Table 4 highlights not only the role of the MW model but also the extreme-valued nature of the distributions. In fact, it can be seen that apart from the percentage of weights set to zero, the vast majority of the remaining weights have a value of either P or $-P$, with less than 1.67% of the weights attaining one of the other intermediate values. We remark that this type of phenomenon is recurrent in the literature under the name of magnitude pruning; see Han et al. (2015), Morcos et al. (2019), and Blalock et al. (2020). In our setting, such a phenomenon occurs spontaneously.

5.9. Experiment 6

The goal of the sixth experiment is to study the impact of the multiobjective model and the weight distributions over a different data set, namely the Heart Disease data set by Janosi et al. (1988).

Table 5 reports the average accuracy and weight distributions of this experiment. The average was performed over five instances, and for each instance, 200 data samples were used, where 80% was used for the training and

Figure 6. (Color online) Comparison of Accuracy for Different Values of P



Note. Note how using an exponentially larger research space, namely using values of P greater than one, does not improve the accuracy.

Table 4. Weight Distributions of the INNs Whose Accuracy Is Depicted in Figure 6

Value of P	Images per class	$w = -P$	$w = 0$	$w = P$	Others
1	2	4.06	73.66	22.28	—
	6	4.75	73.70	21.55	—
	10	5.15	74.02	20.83	—
	20	7.12	61.31	31.57	—
	30	6.21	74.89	18.90	—
	40	13.16	64.98	21.86	—
3	2	3.86	75.91	20.15	0.08
	6	4.60	73.46	21.63	0.31
	10	6.25	73.50	19.72	0.53
	20	12.48	69.58	17.19	0.75
	30	13.06	74.00	11.89	1.05
	40	13.99	65.51	19.31	1.19
7	2	3.90	75.89	20.10	0.11
	6	4.46	73.40	21.76	0.38
	10	8.82	68.00	22.60	0.58
	20	12.46	65.56	21.06	0.92
	30	14.99	73.82	9.95	1.24
	40	18.05	70.90	9.40	1.65
15	2	3.69	75.89	20.30	0.12
	6	4.68	73.31	21.64	0.37
	10	5.53	73.28	20.61	0.58
	20	7.43	69.34	22.19	1.04
	30	13.76	67.64	17.23	1.37
	40	17.34	70.86	10.13	1.67

Notes. The column $w = -P$ indicates the percentage of weights that are equal to $-P$ and so on. The networks have different values but similar extremal weight distributions, where less than 2% of the weights attain a value that is not P, $-P$, or zero, indicated as *others*.

20% was used for the test. All the networks have the same architecture, namely [13, 5, 1], and each network's imposed time limit is 60 minutes, equally distributed in the steps of each model: 60 minutes for the `SM` model, 30 minutes + 30 minutes for the `SM+MM` model, and 20 minutes + 20 minutes + 20 minutes for the `SM+MM+MW` model.

Table 5. Average Accuracy and Weight Distribution for the Heart Disease Data Set

Models	Value of P	Average accuracy	$w = -P$	$w = 0$	$w = P$	Others
`SM`	1	74.00	35.43	45.14	19.43	—
	3	75.00	21.43	27.43	16.57	34.57
	7	**77.00**	27.43	14.86	16.86	40.85
	15	77.00	18.86	11.71	12.86	56.57
`SM+MM`	1	75.00	14.57	18.29	67.14	—
	3	75.50	10.57	13.43	63.14	12.86
	7	73.50	12.29	9.43	52.57	25.71
	15	**78.50**	11.71	5.71	46.86	35.72
`SM+MM+MW`	1	**75.50**	9.43	27.43	63.14	—
	3	**76.00**	5.43	25.43	57.43	11.71
	7	73.50	7.43	18.57	49.71	24.29
	15	**78.50**	8.57	13.43	45.71	32.29

Notes. The column average accuracy depicts the average percentage of correctly classified test data. The column $w = -P$ indicates the percentage of weights that are equal to $-P$ and so on. For each value of P, the best result in terms of accuracy with respect to the model is in bold. Notice that in the majority of the cases, the multiobjective function performs better than the single-objective one. Notice also that even if the percentage of the nonzero and nonextremal weights, indicated as *others*, is higher than the one obtained with the MNIST data set, the distribution is still not uniform.

6. Conclusion and Future Work

This paper introduced the *BeMi* ensemble, a structured architecture of INNs for classification tasks. Each network specializes in distinguishing between pairs of classes and combines different approaches already existing in the literature to preserve feasibility while being robust and simple. These features and the nature of the parameters are critical to enabling neural networks to run on low-power devices. In particular, BNNs can be implemented using Boolean operations and do not need GPUs to run.

The output of the *BeMi* ensemble is chosen by a majority voting system that generalizes the one-versus-one scheme. Notice that the *BeMi* ensemble is a general architecture that could be employed using other types of neural networks.

An interesting conclusion from our computational experiments is a counterintuitive result: that the greater flexibility in the search space of INNs does not necessarily result in better classification accuracy compared with BNNs. We find it noteworthy that our computational evidence supports the idea that simpler BNNs are either superior or equal to INNs in terms of accuracy.

A current limitation of our approach is the strong dependence on the randomly sampled data used for training. In future work, we plan to improve the training data selection by using a k-medoids approach, dividing all images of the same class into disjoint nonempty subsets and considering their centroids as training data. This approach should mitigate the dependency on the sampled training data points.

Second, we also plan to better investigate the scalability of our method with respect to the number of classes of the classification problem training fewer BNNs, namely one for every $\mathcal{J} \in \mathcal{Q} \subset \mathcal{P}$, with $|\mathcal{Q}| \ll |\mathcal{P}|$. Besides the data sets that we exploited, in the future, we intend to investigate data sets more appropriate for the task of few-shot learning (Brigato et al. 2022).

Third, another possible future research direction is to exploit the generalization of our ensemble, allowing us to have networks that distinguish between m classes instead of two, where the total number of classes of the problem is $n \gg m$. Note that the definitions of this generalization are presented in Appendix A.

Fourth, an interesting future research direction is stochastic/robust optimization and scenario generation in the following sense. In the case of MNIST/Fashion-MNIST, for example, the images can be seen as samples of many unknown probability distributions, one for each class; if one would like to train a neural network with an MIP using few images, the selection of these samples with which the training is performed is crucial, and so, the study of this problem from a stochastic point of view could lead to interesting results.

Acknowledgments

The authors thank the anonymous reviewers for their comments. Additionally, the authors thank the participants of the Dagstuhl Seminar 22431 "Data-Driven Combinatorial Optimisation," and the authors also thank Tómas Þorbjarnarson.

Appendix A. Generalization of the Ensemble

The definition of the ensemble introduced in Section 4.2 can be generalize as follows. Define $\mathcal{P}(\mathcal{I})_m$ as the set of all the subsets of the set $\mathcal{I}$ that have cardinality m, where $\mathcal{I}$ is the set of the classes of the classification problem. Then, our structured ensemble is constructed in the following way.

1. We set a parameter $1 < m \leq n = |\mathcal{I}|$.
2. We train an INN denoted by $\mathcal{N}_{\mathcal{J}}$ for every $\mathcal{J} \in \mathcal{P}(\mathcal{I})_m$.
3. When testing a data point, we feed it to our list of trained INNs, namely $(\mathcal{N}_{\mathcal{J}})_{\mathcal{J}\in\mathcal{P}(\mathcal{I})_m}$, obtaining a list of predicted labels $(\mathfrak{e}_{\mathcal{J}})_{\mathcal{J}\in\mathcal{P}(\mathcal{I})_m}$.
4. We then apply a majority voting system.

Note that we set $m > 1$; otherwise, our structured ensemble would have been meaningless. Whenever $m = n$, our ensemble is made of one single INN. When $m = 2$, we are using the one-versus-one scheme presented in the paper.

The idea behind this structured ensemble is that given an input $\boldsymbol{x}^k$ labeled l $(= y^k)$, the input is fed into $\binom{n}{m}$ networks, where $\binom{n-1}{m-1}$ of them are trained to recognize an input with label l. If all of the networks correctly classify the input $\boldsymbol{x}^k$, then at most $\binom{n-1}{m-1} - \binom{n-2}{m-2}$ other networks can classify the input with a different label. With this approach, if we plan to use $r \in \mathbb{N}$ inputs for each label, we are feeding our INNs a total of $m \cdot r$ inputs instead of feeding $n \cdot r$ inputs to a single large INN. When $m \ll n$, it is much easier to train our structured ensemble of INNs rather than training one large INN.

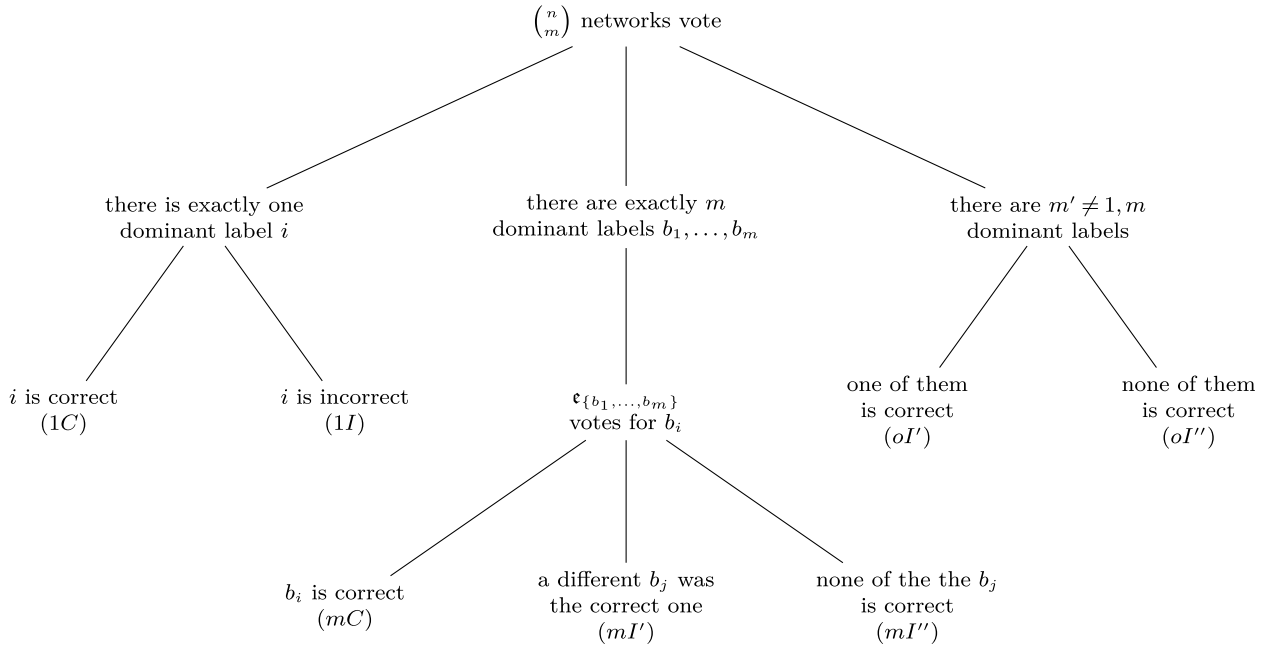
A.1. Majority Voting System.

After the training, we feed one input $\boldsymbol{x}^k$ to our list of INNs, and we need to elaborate on the set of outputs.

Definition A.1 (Dominant Label). For every $b \in \mathcal{I}$, we define

$$C_b = \{\mathcal{J} \in \mathcal{P}(\mathcal{I})_m \,|\, \mathfrak{e}_{\mathcal{J}} = b\},$$

Figure A.1. Tree Diagram of Generalized Label Statuses



and we say that a label b is a *dominant label* if $|C_b| \geq |C_l|$ for every $l \in \mathcal{I}$. We then define the set of dominant labels

$$\mathcal{D} := \{b \in \mathcal{I} \,|\, b \text{ is a dominant label}\}.$$

Using this definition, we can have three possible outcomes.

a. There exists a label $b \in \mathcal{I}$ such that $\mathcal{D} = \{b\} \Rightarrow$ our input is labeled as b.

b. There exist $b_1, \ldots, b_m \in \mathcal{I}$, $b_i \neq b_j$ for all $i \neq j$ such that $\mathcal{D} = \{b_1, \ldots, b_m\}$, so $\mathcal{D} \in \mathcal{P}(\mathcal{I})_m \Rightarrow$ our input is labeled as $\mathfrak{e}_{\{b_1,\ldots,b_m\}} = \mathfrak{e}_{\mathcal{D}}$.

c. $|\mathcal{D}| \neq 1 \wedge |\mathcal{D}| \neq m \Rightarrow$ our input is labeled as $z \notin \mathcal{I}$.

Although case (a) is straightforward, we can label our input even when we do not have a clear winner: that is, when we have trained an INN on the set of labels that are the most frequent (i.e., case (b)). Note that the proposed structured ensemble alongside its voting scheme can also be exploited for regular NNs.

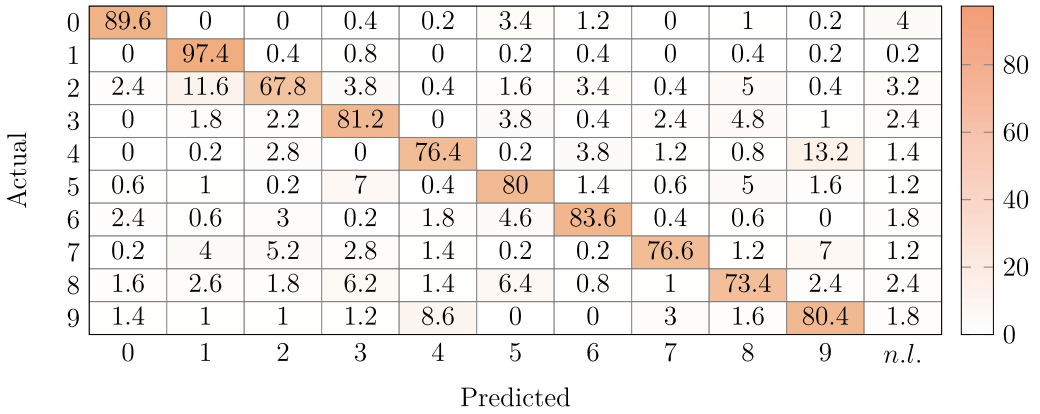
Definition A.2 (Label Statuses). In our labeling system, when testing an input, seven different cases, herein called *label statuses*, can arise. The status names are of the form "number of the dominant labels + fairness of the prediction." The first parameter can be $1, m$, or o, where o means "other cases." The fairness of the prediction is C when it is correct or I when it is incorrect. The subscripts related to I' and I'' only distinguish between different cases. These cases are described through the following tree diagram (Figure A.1).

The reason behind this generalization is the following. Suppose we have trained an NN to distinguish between three different classes: c_1, c_2, c_3. If one class is added to the problem, namely c_4, instead of discarding the trained NN, one could train three other NNs, namely the one that distinguishes between c_1, c_2, c_4; another one for c_1, c_3, c_4; and the last one for c_2, c_3, c_4, and use the majority voting scheme. Note that the training of the smaller networks is linked to smaller MILPs; in fact, if we plan to use 10 input data for each class, we need 40 input data in total, but every network is fed with only 30 of them. Moreover, the training of the three additional NNs can be done in parallel, saving computational time.

Appendix B. Complement to Experiment 1

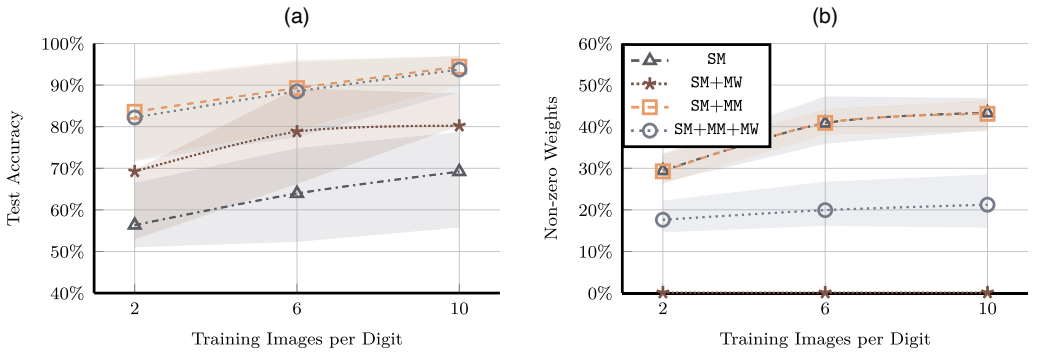
We first performed an experiment with the *BeMi* ensemble trained with 40 images per digit, and we reported the results in a confusion matrix, depicted in Figure B.1. This gave us an idea of what digits were easy or difficult to distinguish between. We then replicate Experiment 1 with a different couple of digits, namely one and eight, which are easier to distinguish than some other digits, like four and nine. Results are depicted in Figure B.2. We can draw the same conclusions of Experiment 1.

Figure B.1. (Color online) Confusion Matrix of the *BeMi* Ensemble Trained with 40 Images per Digits with the Architecture [784, 4, 4, 1]



Notes. We reported the percentages for each couple. Note that the *n.l.* indicates the unclassified images.

Figure B.2. (Color online) Ablation Study with a Different Couple of Digits, Namely 1 and 8, Which Are Easier to Distinguish



Notes. (a) Test accuracy of different models. (b) Simplicity of different models.

Endnote

[1] Acronym from the last names of the two young authors who had this intuition.

References

Anderson R, Huchette J, Ma W, Tjandraatmadja C, Vielma JP (2020) Strong mixed-integer programming formulations for trained neural networks. *Math. Programming* 183(1):3–39.

Banner R, Hubara I, Hoffer E, Soudry D (2018) Scalable methods for 8-bit training of neural networks. *Adv. Neural Inform. Processing Systems* 31:5145–5153.

Bengio Y, Lodi A, Prouvost A (2021) Machine learning for combinatorial optimization: A methodological tour d'horizon. *Eur. J. Oper. Res.* 290(2):405–421.

Bergman D, Huang T, Brooks P, Lodi A, Raghunathan AU (2022) Janos: An integrated predictive and prescriptive modeling framework. *INFORMS J. Comput.* 34(2):807–816.

Bernardelli AM, Gualandi S, Lau HC, Milanesi S (2023) The BeMi stardust: A structured ensemble of binarized neural networks. *Internat. Conf. Learn. Intelligent Optimization (LION)* (Springer, Cham, Switzerland), 443–458.

Bernardelli AM, Gualandi S, Milanesi S, Lau HC, Yorke-Smith N (2024) Multi-objective linear ensembles for robust and sparse training of few-bit neural networks. http://dx.doi.org/10.1287/ijoc.2023.0281.cd, https://github.com/INFORMSJoC/2023.0281.

Bishop CM, Nasrabadi NM (2006) *Pattern Recognition and Machine Learning* (Springer, New York).

Blalock DW, Gonzalez Ortiz JJ, Frankle J, Guttag JV (2020) What is the state of neural network pruning? *Proc. Machine Learn. Systems*, 129–146.

Blott M, Halder L, Leeser M, Doyle L (2019) Qutibench: Benchmarking neural networks on heterogeneous hardware. *ACM J. Emerging Tech. Comput. Systems* 15(4):1–38.

Botoeva E, Kouvaros P, Kronqvist J, Lomuscio A, Misener R (2020) Efficient verification of ReLu-based neural networks via dependency analysis. *Proc. Conf. AAAI Artificial Intelligence* 34(4):3291–3299.

Brigato L, Barz B, Iocchi L, Denzler J (2022) Image classification with small datasets: Overview and benchmark. *IEEE Access* 10(2022):49233–49250.

Cai J, Nguyen KN, Shrestha N, Good A, Tu R, Yu X, Zhe S, Serra T (2023) Getting away with more network pruning: From sparsity to geometry and linear regions. *Internat. Conf. Integration Constraint Programming Artificial Intelligence Oper. Res. (CPAIOR)* (Springer, Cham, Switzerland), 200–218.

Cappart Q, Chételat D, Khalil EB, Lodi A, Morris C, Veličković P (2023) Combinatorial optimization and reasoning with graph neural networks. *J. Machine Learn. Res.* 24(130):1–61.

ElAraby M, Wolf G, Carvalho M (2023) OAMIP: Optimizing ANN architectures using mixed-integer programming. *Internat. Conf. Integration Constraint Programming Artificial Intelligence Oper. Res. (CPAIOR)* (Springer, Berlin, Heidelberg), 219–237.

Fischetti M, Jo J (2018) Deep neural networks and mixed integer linear optimization. *Constraints* 23(3):296–309.

Gholami A, Kim S, Dong Z, Yao Z, Mahoney MW, Keutzer K (2022) A survey of quantization methods for efficient neural network inference. Thiruvathukal GK, Lu Y-H, Kim J, Chen Y, Chen B, eds. *Low-Power Computer Vision* (Chapman and Hall/CRC, New York), 291–326.

Good A, Lin J, Yu X, Sieg H, Fergurson M, Zhe S, Wieczorek J, Serra T (2022) Recall distortion in neural network pruning and the undecayed pruning algorithm. *Adv. Neural Inform. Processing Systems* 35:32762–32776.

Gurobi Optimization LLC (2023) Gurobi Optimizer reference manual. Accessed August 9, 2023, https://www.gurobi.com.

Han S, Mao H, Dally WJ (2015) Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. Preprint, submitted October 1, https://arxiv.org/abs/1510.00149.

Huang T, Ferber AM, Tian Y, Dilkina B, Steiner B (2023) Searching large neighborhoods for integer linear programs with contrastive learning. *Proc. 40th Internat. Conf. Machine Learn.*, vol. 202 (PMLR, New York), 13869–13890.

Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y (2016) Binarized neural networks. *Adv. Neural Inform. Processing Systems* 29:4107–4115.

Huchette J, Muñoz G, Serra T, Tsay C (2023) When deep learning meets polyhedral theory: A survey. Preprint, submitted April 29, https://arxiv.org/abs/2305.00241.

Janosi A, Steinbrunn W, Pfisterer M, Detrano RU (1988) Heart disease data set. Accessed August 9, 2023, http://archive.ics.uci.edu/ml/datasets/Heart+Disease.

Jiang Y, Krishnan D, Mobahi H, Bengio S (2019) Predicting the generalization gap in deep networks with margin distributions. *Internat. Conf. Learn. Representations (ICLR)* (OpenReview.net).

Kawaguchi K, Kaelbling LP, Bengio Y (2017) Generalization in deep learning. Preprint, submitted October 16, https://arxiv.org/abs/1710.05468.

Keskar NS, Mudigere D, Nocedal J, Smelyanskiy M, Tang PTP (2017) On large-batch training for deep learning: Generalization gap and sharp minima. *Internat. Conf. Learn. Representations (ICLR)*, vol. 5 (OpenReview.net).

Khalil EB, Gupta A, Dilkina B (2019) Combinatorial attacks on binarized neural networks. *Internat. Conf. Learn. Representations (ICLR)* (OpenReview.net).

Kurtz J, Bah B (2021) Efficient and robust mixed-integer optimization methods for training binarized deep neural networks. Preprint, submitted October 21, https://arxiv.org/abs/2110.11382.

LeCun Y, Bengio Y, Hinton G (2015) Deep learning. *Nature* 521(7553):436–444.

LeCun Y, Cortes C, Burges CJ (1998) The MNIST database of handwritten digits. Accessed August 9, 2023, http://yann.lecun.com/exdb/mnist.

Li X, Sun C, Ye Y (2020) Simple and fast algorithm for binary integer and online linear programming. *Adv. Neural Inform. Processing Systems* 33:9412–9421.

Lin X, Zhao C, Pan W (2017) Toward accurate binary convolutional neural network. *Adv. Neural Inform. Processing Systems* 30:345–353.

Lombardi M, Milano M (2018) Boosting combinatorial problem modeling with machine learning. Preprint, submitted July 15, https://arxiv.org/abs/1807.05517.

Mistry M, Letsios D, Krennrich G, Lee RM, Misener R (2021) Mixed-integer convex nonlinear optimization with gradient-boosted trees embedded. *INFORMS J. Comput.* 33(3):1103–1119.

Moody J (1991) The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. *Adv. Neural Inform. Processing Systems* 4:847–854.

Morcos A, Yu H, Paganini M, Tian Y (2019) One ticket to win them all: Generalizing lottery ticket initializations across datasets and optimizers. *Adv. Neural Inform. Processing Systems* 32:4932–4942.

Neyshabur B, Bhojanapalli S, McAllester D, Srebro N (2017) Exploring generalization in deep learning. *Adv. Neural Inform. Processing Systems* 30:5947–5956.

Patil V, Mintz Y (2022) A mixed-integer programming approach to training dense neural networks. Preprint, submitted January 3, https://arxiv.org/abs/2201.00723.

Roger C, Molina R, Rey C, Serra T, Puertas E, Pujol O (2022) Training thinner and deeper neural networks: Jumpstart regularization. Schaus P, ed. *Integration Constraint Programming Artificial Intelligence Oper. Res. CPAIOR 2022*, Lecture Notes in Computer Science, vol. 13292 (Springer, Cham, Switzerland), 345–357.

Sakr C, Choi J, Wang Z, Gopalakrishnan K, Shanbhag N (2018) True gradient-based training of deep binary activated neural networks via continuous binarization. *2018 IEEE Internat. Conf. Acoustics Speech Signal Processing (ICASSP)* (IEEE, Piscataway, NJ), 2346–2350.

Serra T, Kumar A, Ramalingam S (2020) Lossless compression of deep neural networks. Hebrard E, Musliu N, eds. *Integration Constraint Programming Artificial Intelligence Oper. Res. CPAIOR 2020*, Lecture Notes in Computer Science, vol. 12296 (Springer, Cham, Switzerland), 417–430.

Serra T, Yu X, Kumar A, Ramalingam S (2021) Scaling up exact neural network compression by ReLU stability. *Adv. Neural Inform. Processing Systems* 34:27081–27093.

Tang W, Hua G, Wang L (2017) How to train a compact binary neural network with high accuracy? *Thirty-First AAAI Conf. Artificial Intelligence* (AAAI Press, Palo Alto, CA), 2625–2631.

Thorbjarnarson T, Yorke-Smith N (2023) Optimal training of integer-valued neural networks with mixed integer programming. *PLoS One* 18(2):e0261029.

Tjandraatmadja C, Anderson R, Huchette J, Ma W, Patel KK, Vielma JP (2020) The convex relaxation barrier, revisited: Tightened single-neuron relaxations for neural network verification. *Adv. Neural Inform. Processing Systems* 33:21675–21686.

Tjeng V, Xiao KY, Tedrake R (2018) Evaluating robustness of neural networks with mixed integer programming. *Internat. Conf. Learn. Representations* (OpenReview.net).

Toro Icarte R, Illanes L, Castro MP, Cire AA, McIlraith SA, Beck JC (2019) Training binarized neural networks using MIP and CP. *Internat. Conf. Principles Practice Constraint Programming*, vol. 11802 (Springer, Cham, Switzerland), 401–417.

Tsay C, Kronqvist J, Thebelt A, Misener R (2021) Partition-based formulations for mixed-integer optimization of trained ReLU neural networks. *Adv. Neural Inform. Processing Systems* 34:3068–3080.

Vanschoren J (2019) Meta-learning. Hutter F, Kotthoff L, Vanschoren J, eds. *Automated Machine Learning*, Springer Series on Challenges in Machine Learning (Springer, Cham, Switzerland), 35–61.

Wang K, Lozano L, Cardonha C, Bergman D (2023) Optimizing over an ensemble of trained neural networks. *INFORMS J. Comput.* 35(3):652–674.

Williams HP (2013) *Model Building in Mathematical Programming* (John Wiley & Sons, Chichester, UK).

Xiao H, Rasul K, Vollgraf R (2017) Fashion-mnist: A novel image data set for benchmarking machine learning algorithms. Preprint, submitted August 25, https://arxiv.org/abs/1708.07747.

Ye H, Xu H, Wang H, Wang C, Jiang Y (2023) GNN&GBDT-guided fast optimizing framework for large-scale integer programming. *Proc. 40th Internat. Conf. Machine Learn.*, vol. 202 (PMLR, New York), 39864–39878.

Young HP (1988) Condorcet's theory of voting. *Amer. Political Sci. Rev.* 82(4):1231–1244.

Yu X, Serra T, Ramalingam S, Zhe S (2022) The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. *Internat. Conf. Machine Learn. (ICML)* (PMLR, New York), 25668–25683.