Presented at: IJCAI-PRICAI'20 Workshop on Data Science Meets Optimisation, Yokohama, Japan, January 2021.

On Training Neural Networks with Mixed Integer Programming

Tómas Þorbjarnarson , Neil Yorke-Smith*
Delft University of Technology, The Netherlands
T.orbjarnarson@student.tudelft.nl, n.yorke-smith@tudelft.nl

Abstract

Recent work has shown potential in using Mixed Integer Programming (MIP) solvers to optimize certain aspects of neural networks (NN). However little research has gone into training NNs with solvers. State of the art methods to train NNs are typically gradient-based and require significant data, computation on GPUs and extensive hyper-parameter tuning. In contrast, training with MIP solvers should not require GPUs or hyper-parameter tuning but can likely not handle large amounts of data. This work builds on recent advances that train binarized NNs using MIP solvers. We go beyond current work by formulating new MIP models to increase the amount of data that can be used and to train non-binary integer-valued networks. Our results show that comparable results to using gradient descent can be achieved when minimal data is available.

1 Introduction

Training NNs using gradient-based optimization methods can be tedious. Hyper-parameters require meticulous and computationally-intensive tuning to reach the best results. Although state-of-the-art methods use gradient-based optimization, these standard methods often require a large number of neurons and immense amounts of data.

A number of studies have been performed recently that try to counteract the trend of increasingly-large networks. For instance, a branch of NN optimization that intends to reduce the size of networks has gained some traction [Huang *et al.*, 2020]. The motivation of these studies is to decrease the memory needs of NNs and to increase efficiency in training and using them, without degrading the networks' generalization ability.

We posit that modelling NNs using mixed integer programming (MIP) and training them with discrete optimisation solvers could work well in reduced memory settings. Recent work shows that this idea is feasible when using minimal data [Icarte *et al.*, 2019]. It is unclear whether MIP solvers can perform well with large networks and large amounts of data. Instead, we see potential in using solvers to train smaller networks with small batches of data.

Training with solvers may not compare to gradient-based methods at large scale, but MIP-based training will reduce hyper-parameter tuning considerably. Choosing learning rates, momentum, decay, the number of epochs, batch sizes and more will be unnecessary. By modelling NNs using MIP and reasonable objective functions, the solver can in principle find a guaranteed optimal solution. This is advantageous, as even after extensive hyper-parameter tuning for gradient-based methods, it can be unclear whether an optimal NN configuration has been reached.

Previous work has shown the advantages of using MIP to solve particular aspects of NNs. Fischetti and Jo [2018] show that it is feasible to model NNs using MIP and use solvers to generate optimized adversarial images for the network. Icarte *et al.* [2019] show the feasibility of directly training binary NNs (BNN) using MIP solvers. In this paper, we go beyond Icarte *et al.* [2019] to train integer-valued NNs using MIP models. Further, we provide new models that resemble popular loss functions from gradient-based optimization.

This work-in-progress contributes an expandable framework to train NNs using MIP solvers. This framework provides flexibility such that the user can choose the range of the networks parameters accordingly to adjust the memory usage of the NN. Our proposed models can perform comparably to gradient-based methods when minimal data is available. This can be useful for training on small datasets, and it reduces the need for hyper-parameter tuning and the practical necessity of using GPUs to train NNs.

The paper is structured as follows. Section 2 describes our approach. Section 3 presents results of two experiments. Section 4 positions our contribution in the literature. Section 5 concludes the paper by identifying future directions.

2 Modelling Approach

Icarte *et al.* [2019] find that their MIP models are limited by the amount of data they can feasibly use to train NNs. We aim to increase the amount of data and provide models that perform similarly to a gradient descent baseline, given a limited amount of data on which to train. Further, we aim to capture a range of NN loss functions.

To this end, we propose three novel NN MIP models that have the same base model but separate objective functions to

*Corresponding author

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

$$(u_{\ell j}^k = 1) \Longrightarrow (\sum_{i \in N_{\ell-1}} c_{i\ell j}^k + b_{\ell j} \geq 0) \qquad \forall \ell \in \mathcal{L}^{L-1}, j \in N_\ell, k \in T \tag{2}$$

$$(u_{\ell j}^k = 0) \Longrightarrow (\sum_{i \in N_{\ell-1}} c_{i\ell j}^k + b_{\ell j} \leq -\epsilon) \qquad \forall \ell \in \mathcal{L}^{L-1}, j \in N_\ell, k \in T \tag{3}$$

$$c_{i1j}^k = x_i^k \cdot w_{i1j} \qquad \forall i \in N_0, j \in N_1, k \in T \tag{4}$$

$$c_{i\ell j}^k - w_{i\ell j}^k + 2P \cdot u_{(\ell-1)i}^k \leq 2P \qquad \forall \ell \in \mathcal{L}_2 i \in N_{\ell-1}, j \in N_\ell, k \in T \tag{5}$$

$$c_{i\ell j}^k + w_{i\ell j}^k - 2P \cdot u_{(\ell-1)i}^k \leq 0 \qquad \forall \ell \in \mathcal{L}_2 i \in N_{\ell-1}, j \in N_\ell, k \in T \tag{6}$$

$$c_{i\ell j}^k - w_{i\ell j}^k - 2P \cdot u_{(\ell-1)i}^k \geq -2P \qquad \forall \ell \in \mathcal{L}_2 i \in N_{\ell-1}, j \in N_\ell, k \in T \tag{7}$$

$$c_{i\ell j}^k + w_{i\ell j}^k + 2P \cdot u_{(\ell-1)i}^k \geq 0 \qquad \forall \ell \in \mathcal{L}_2 i \in N_{\ell-1}, j \in N_\ell, k \in T \tag{8}$$

$$w_{i\ell j}, b_{Lj} \in \{-P, ..., P\} \qquad \forall \ell \in \mathcal{L}, i \in N_{\ell-1}, j \in N_\ell \tag{9}$$

$$c_{i\ell j}^k \in \mathbb{R} \qquad \forall \ell \in \mathcal{L}, i \in N_{\ell-1}, j \in N_\ell, k \in T \tag{10}$$

$$u_{\ell j}^k \in \{0, 1\} \qquad \forall \ell \in \mathcal{L}^{L-1}, j \in N_\ell, k \in T \tag{11}$$

$$P \in \mathcal{Z}^+ \tag{12}$$

serve different purposes. The base model they share in common is a slightly modified model from Icarte *et al.* [2019]. This base NN MIP model captures: a multi-layer perceptron with sign activation function. Our modifications include removing any objective function and constraints specific to Icarte *et al.*'s methodology. We also modify constraints to allow the network's parameters to take larger ranges. The remaining constraints ensure that the NN's calculations are correct.

The decision variables to optimize are the network's integer weights and biases (9), continuous connections (10) and binary activations (11). There are L layers in the network. We specify two sets to simplify notation. The layer sets are $\mathcal{L} = \{2, ..., L\}$ and $\mathcal{L}^{L-1} = \{1, ..., L-1\}$. The number of neurons per layer ℓ is defined as N_ℓ.

The variable $w_{i\ell j}$ is the weight of the connection from neuron $i \in N_{\ell-1}$ to neuron $j \in N_\ell$. The variable $b_{\ell j}$ is the bias of neuron $j \in N_\ell$. To model inter-layer calculations for each sample, we use the variable $c_{i\ell j}^k$. The binary variable $u_{\ell j}^k$ models the activation of neuron $j \in N_\ell$ for every sample $k \in T$. With it we model the sign activation function: if the input to neuron $j \in N_\ell$ is negative, $u_{\ell j}^k = 0$ (3), otherwise $u_{\ell j}^k = 1$ (2). Subsequently, $c_{i\ell j}^k$ calculates using $u_{\ell j}^k$. To properly model the sign function, the values {0,1} are mapped to {-1,1}. Thus, in following layers, equations (5–8) ensure that $c_{i\ell j}^k = (2u_{(\ell-1)i}^k - 1) \cdot w_{i\ell j}$. Finally, $\hat{y}_j^k$ models the normalized value in output neuron $j \in N_L$ for sample k. We choose $\epsilon = 1 \cdot 10^-5$ in equation (3) to model the inequality in accordance with the variable precision tolerance of the solver we will use [Gurobi Optimization, 2019].

Equation (4) models calculations of the NN's first layer while equations (5–8) model the subsequent layers as noted. Equation (1) calculates the values in neurons in the final layer. The value $\hat{y}_j^k$ therefore represents an encoded predicted value of the network.

All sample labels are encoded using +1/-1 encoding in accordance with the theme of BNNs and using the sign function as an activation function. Further, the output neurons (1) are normalized using a linear approximation method such that all values are approximately between -1 and 1. In a MIP model we cannot use non-linear normalization functions, such as softmax or sigmoid, so we approximately linearly normalize instead.

We next explain our three model variants and their objective functions.

2.1 Model 1: Max-correct

Our first proposed model, max-correct, aims to maximize the number of correct predictions of training samples. It uses a binary variable for each output neuron to denote whether the sample has the correct label.

$$\max \sum_{k \in T} \sum_{j \in N_L : y_j^k = 1} o_j^k \tag{13}$$

$$(o_j^k = 1) \Longrightarrow (\hat{y}_j^k \geq 0) \qquad \forall j \in N_L, k \in T \tag{14}$$

$$(o_j^k = 0) \Longrightarrow (\hat{y}_j^k \leq -\epsilon) \qquad \forall j \in N_L, k \in T \tag{15}$$

$$\sum_{k \in T} o_j^k = 1 \qquad \forall j \in N_L \tag{16}$$

$$o_j^k \in \{0, 1\} \qquad \forall j \in N_\ell, k \in T \tag{17}$$

This model is simple and fast. It requires only one output neuron per sample to be positive and maximizes the number of positive output neurons that correspond to the correct label. However, there is little confidence in predictions as they just barely need to be correct. Therefore, similar samples in the testing dataset may be incorrectly classified.

2.2 Model 2: Min-hinge

To increase the confidence of predictions, we propose our second model, min-hinge. This model is inspired by the squared hinge loss (18) that can be used when using +1/-1 encoding for labels. The loss function is thus:

$$L = \sum_j \max\left(0, \frac{1}{2} - \hat{y}^{(j)} y^{(j)}\right)^2 \tag{18}$$

The squared hinge loss function is non-linear but can be approximated using piecewise linear (PWL) functions. PWL functions are defined by a number of break points and lines between the break points. By choosing sufficient break points, the non-linear squared hinge loss function can be approximated. We can then simply input the multiplication of our predicted value $\hat{y}_j^k$ (1) with the encoded label y_j^k to calculate the loss for a single output neuron for a single sample.

The total loss to be minimized is the sum over all output neurons for all samples (19):

$$\min \sum_{k \in T} \sum_{j \in N_L} f(\hat{y}_j^k \cdot y_j^k) \tag{19}$$

The advantages of this model are that predictions are pushed to be more confident. The max-correct (2.1) model's target is to maximize the number of correct predictions. The min-hinge model also aspires to do so, but additionally aims to make each prediction to be above the margin of $\frac{1}{2}$. The squared hinge loss (18) is taken from literature so we can be more sure of the network being optimized reasonably.

2.3 Model 3: Sat-margin

Our final proposed model, sat-margin, combines aspects from the previous two models. It optimizes a sum of binary variables, like max-correct (2.1), but also aims to confidently predict each sample, like min-hinge (2.2).

$$max \sum_{k \in T} \sum_{j \in N_L} o_j^k \tag{20}$$

$$(o_j^k = 1) \implies (\hat{y}_j^k \cdot y_j^k \geq m) \qquad \forall j \in N_L, k \in T \tag{21}$$

$$(o_j^k = 0) \implies (\hat{y}_j^k \cdot y_j^k \leq m - \epsilon) \quad \forall j \in N_L, k \in T \tag{22}$$

$$o_j^k \in \{0, 1\} \qquad \forall j \in N_\ell, k \in T \tag{23}$$

$$m \in \{0, 1\} \tag{24}$$

The advantages of using the sat-margin model are that it tries to reach the same minimum objective value as min-hinge (2.2). However, there is no need for defining PWL functions. This may help with future work that could have additional objective functions.

3 Experimental Results

We undertake two experiments that train NNs using the proposed models. Experiment 1 compares training BNNs using our three models to a gradient descent (GD) baseline on increasingly large training datasets. Experiment 2 trains integer NNs (INN) using the sat-margin model (2.3).

Both experiments train NNs using the Adult dataset [Kohavi and Becker, 1996]. The associated task with the dataset is binary classification. There are 32560 samples in the training set and 16280 in the testing set. Each sample represents an individual and the corresponding label denotes whether the individual has a yearly income of over 50K or not. Each sample has 14 attributes, 8 of which are categorical and 6 are numerical. We pre-process the data such that the categorical attributes are one-hot encoded and the numerical attributes are normalized to be between 0 and 1.

We use Gurobi Optimizer version 9.0.1 [Gurobi Optimization, 2019] to solve our MIP models. We use a method to train BNNs introduced by Courbariaux and Bengio [2016] as our GD baseline. Experiments are run on an 8-core machine with an Intel Xeon Gold 6148 CPU at 2.40GHz with 32GB RAM. Each run of a model has a maximum time limit of 10 hours.

The networks trained have one hidden layer containing 16 neurons. We use the sign function as our activation function in the hidden layer. Each network has two neurons in the final layer, one for each label the sample can take. To assign a class to the sample, we choose the larger value of the output neurons. If the values are equal for a sample, we choose the label randomly.

To shorten solving time, we do not require each model to be solved to optimality. Reaching the global optimum with limited data will likely lead to over-training as well. We therefore allow the solver to stop optimizing once the network is ensured to have a training accuracy of above 90%.

3.1 Experiment 1

Each network is trained using up to 280 samples. We compare how our models perform compared to GD for such limited data. We are interested in how the resulting networks generalize to the testing set. We also consider how the runtimes of each model change with more training data. The purpose of this experiment is to research the feasibility of using MIP models to train NNs with limited data. We would like to know how the proposed models compare to each other so as to know what future work may be interesting.

We hypothesize that the proposed MIP models will perform comparably to the GD baseline. The min-hinge (2.2) and sat-margin (2.3) models should have similar testing accuracy but should both outperform the max-correct (2.1) model. However, the max-correct model will have a much shorter runtime than min-hinge and sat-margin.

3.2 Experiment 2

We again train NNs using up to 280 samples. In this experiment however, we only use the sat-margin model (2.3). We would like to research the effects of increasing the range of the variables that represent weights and biases in the networks. We compare training BNNs like in Experiment 1, where $P = 1$ (12), to training INNs with $P = 3$,$P = 7$,$P = 15$. Each increase in range represents an extra bit needed to store a network's parameter in memory. The purpose of this experiment is to investigate the benefits of training low-bitwidth INNs instead of BNNs.

We hypothesize that NNs with larger ranges for parameters will be easier to train. Although the increased range results in larger search spaces, it will be easier to reach good solutions. BNNs have very constrained variables and smaller search spaces, but the increased range of INNs could lead to fitting to training samples easier. Because INNs may find better solutions quicker, we hypothesize that they will have shorter runtimes. However, they may not generalize better. Increased ranges of parameters could lead to more over-training, while BNNs are very regularized.

3.3 Results

Results for Experiment 1 can be seen in Figures 2 and 3, and the training seen in Figure 1. Each model is run three times for every amount of training data. Each run uses a random subset of the available data. The figures show the average results over the runs. All runs resulted in training accuracies of above 95%.

Figure 2 shows how well the models generalize and compare them to the GD baseline (*gd_nn*). We see that with more data available, testing accuracy generally increases for every model. The max-correct model is erratic and performs worst of the models. The min-hinge and sat-margin models perform similarly and even slightly outperform GD, with increasing amounts of data.

Figure 3 shows the evolution of runtimes with increasing amounts of data. It becomes clear that with more than 150 samples to train on, the solving time for min-hinge and sat-margin drastically increases. Max-correct and GD, on the other hand, have very short runtimes.

Results for Experiment 2 can be seen in Figures 5 and 6, and the training seen in Figure 4. We use the sat-margin model with P values $P = 1$, $P = 3$, $P = 7$ and $P = 15$. Like in Experiment 1, we run each variation three times for every amount of training data. Each run uses a random subset of the available data. The figures show the average results over the runs. Again, all runs resulted in training accuracies of above 95%.

Figure 5 shows how well the model variations generalize to the testing dataset. When more than 100 examples are available, all variations perform very similarly. The model with the lowest bound, the BNN, generalizes slightly better. Figure 6 shows the difference in runtimes of the variations. It is clear that with more than 150 samples in the training set, higher ranges of parameters manage to solve much quicker.

3.4 Discussion

As hypothesized, the max-correct model is quicker than our other proposed models. However, because it does not require any confidence in predictions, it results in lower testing accuracies. The min-hinge and sat-margin models perform similarly well as we had hoped. They both push predictions to be more confident and therefore generalize better. In Figure 3 we see that sat-margin does take longer to solve than min-hinge. However, we use it in Experiment 2 because it does not require piecewise linear functions. Using sat-margin with more complex Gurobi methods may therefore be easier in future work.

We see that there is a considerable difference in training INNs compared to BNNs. The increased range in parameters allows the model to solve much quicker, without degrading generalization much. There is a trade-off in increasing parameter ranges. With larger ranges, more memory is needed to represent the network. Nevertheless, with the aim of pushing the limits on how much data can be feasibly used to train on, training low-bitwidth INNs is preferable to training BNNs.

4 Related Work

While an emphasis of work at the intersection of operations research and machine learning has been exploiting the latter to help solve optimisations problems studied by the former, an important thrust is also the use of OR models and tools to advance the latter [Bengio *et al.*, 2018].

Fischetti and Jo [2018] researched modelling NNs using MIP to optimize certain aspects of the network. Instead of training using solvers, they use pre-trained networks and use solvers to find optimized adversarial examples. They model the problem to modify examples minimally such as to fool the network into classifying the example incorrectly.

Tjeng *et al.* [2019] go further into evaluating robustness of NNs with MIP by finding optimized adversarial examples. They provide tight formulations for non-linearities in the models which result in considerably quicker solving times. Due to these speedups, the authors manage to solve larger and more complex networks. While Fischetti and Jo [2018] focus on multi-layer perceptrons as we do in our paper, Tjeng *et al.* [2019] examine the robustness of deeper networks as well as networks with convolutional and residual layers.

Anderson *et al.* [2019] provide strong MIP formulations of pre-trained NNs. Like Fischetti and Jo [2018] and Tjeng *et al.* [2019], they model ReLU networks and evaluate robustness of networks by modifying samples with minimal perturbations. However, their model removes the need for additional variables to model the ReLU function. They provide proofs of their strong formulations and demonstrate how the formulations can decrease solving time considerably.

Grimstad and Andersson [2019] similarly research optimizing certain aspects of pre-trained NNs by using MIP: namely, using ReLU networks as surrogate models in MIP. They highlight the importance of bound tightening techniques and how it effects the efficiency of the models. The results show that ReLU networks are suitable as surrogate models in MIP, at least for small, shallow networks. In contrast to these works, however, we directly train NN using MIP.

The closest work to our is Icarte *et al.* [2019], who proceeded to direct train BNNs using MIP models. They provide novel methods to model BNNs. Their models train BNNs while also optimizing certain aspects of them. Instead of optimizing a function that leads to high training accuracy, they introduce constraints that ensure the network fits to training data perfectly. They then propose two variations of the model. Variation 1 maximizes the number of zero-weight connections in the network, thus effectively removing as many unnecessary connections as possible. Variation 2 maximizes margins on every neuron in the network, which should lead

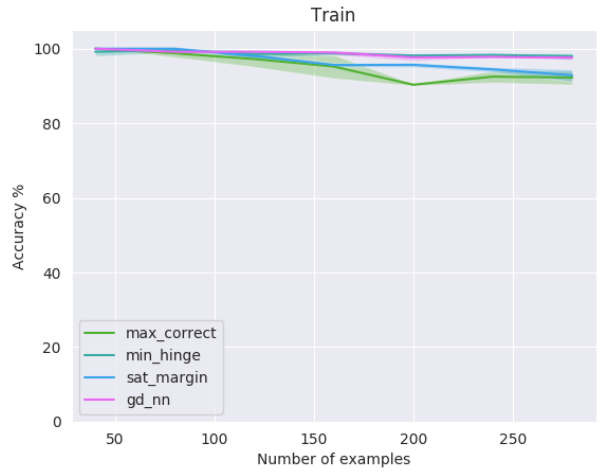

Figure 1: Training accuracy of models for Adult dataset

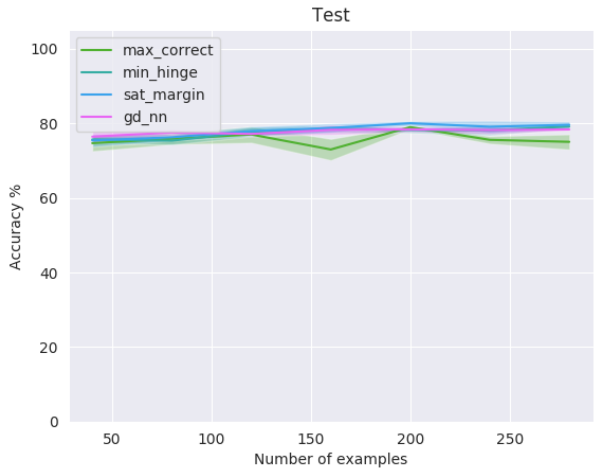

Figure 2: Testing accuracy of models for Adult dataset

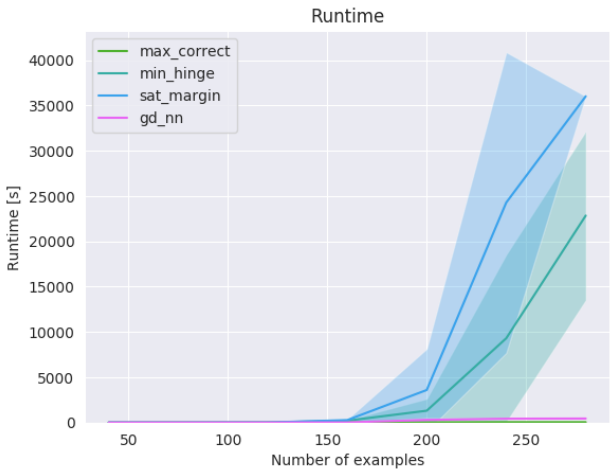

Figure 3: Runtimes of models for Adult dataset

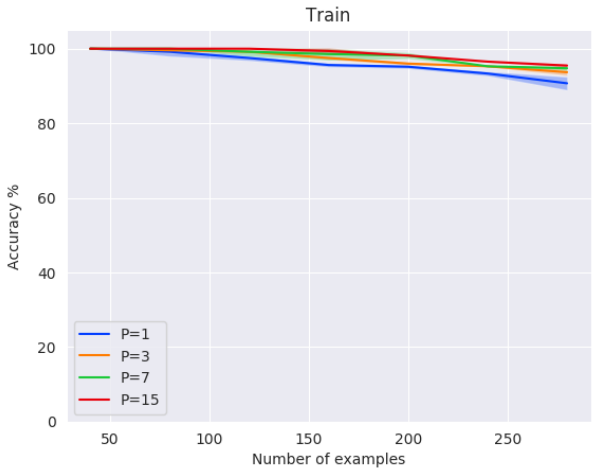

Figure 4: Training accuracy of sat-margin model with different parameter ranges P for Adult dataset

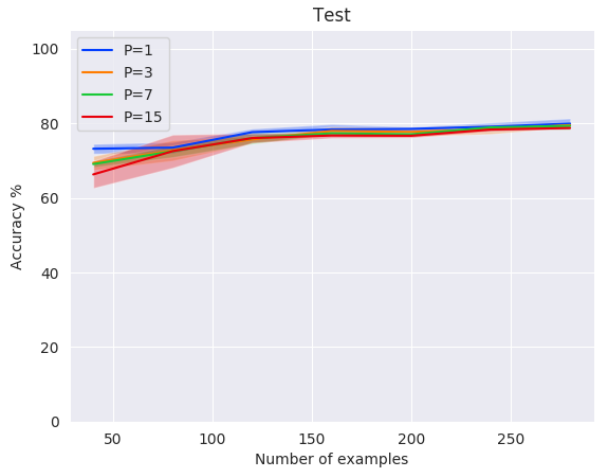

Figure 5: Testing ccuracy of sat-margin model with different parameter ranges P for Adult dataset

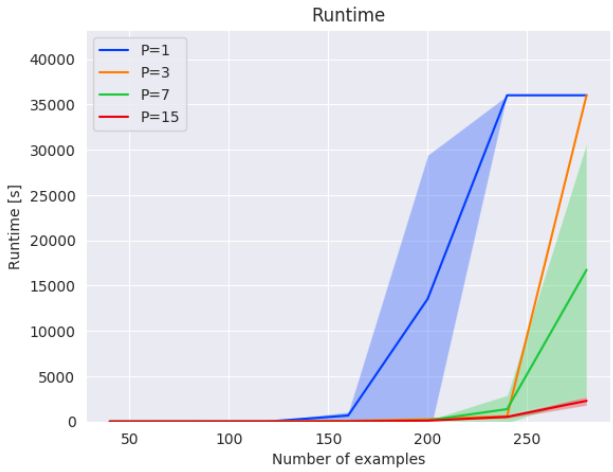

Figure 6: Runtime sat-margin model with different parameter ranges P for Adult dataset

to more confident activations and predictions.

Our work differs from the work by Icarte *et al.* [2019] as our proposed models directly optimize loss functions to maximize accuracy. Further, our models are more relaxed and therefore are more capable of handling more training data. Our work is also more general, in that we are able to handle the important class of non-binary NNs.

5 Conclusion

This paper builds on the recent idea of training neural networks using MIP. We provided a framework that is flexible to change objective functions as well as the range of integers the network can use. Training NNs using our proposed models requires almost no hyper-parameter tuning, in comparison to extensive hyper-parameter tuning needed when using gradient based methods.

Our results to date show that our proposed models can perform comparably to gradient descent baselines when using minimal data to train and relatively small NNs with integer or binary parameters. The results also show that solving time can be considerably improved by allowing NNs to have larger ranges for parameters, in comparison to binary parameters.

Training NNs using MIP is still limited to the amount of training data the models can handle. Nevertheless, our methods have pushed the limits on the amounts of data they can use. It would be interesting to push these limits even further. When using gradient-based training, parameters are often updated on small batches of data. It would be interesting to research a batch training methodology for MIP models.

The NN models used in Grimstad and Andersson [2019] and Fischetti and Jo [2018] use the ReLU activation function while our models use the sign activation function. The ReLU function is more flexible than the sign function. Thus it would be interesting to extend our models to handle the ReLU function, as well as other potential activation functions.

Lastly, in this paper we compare our models to gradient-descent baselines on the Adult dataset. This dataset contains much more training data than our models can handle to date. It would be interesting to exploit our models in combination with more applicable datasets. Datasets with minimal available training data may be an environment where the advantages of training NNs using MIP are most evident.

Acknowledgements

Thanks to S. van der Laan and L. Scavuzzo.

References

Ross Anderson, Joey Huchette, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. In *Proceedings of IPCO'19*, volume 11480 of *Lecture Notes in Computer Science*, pages 27–42. Springer, 2019.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: A methodological tour d'horizon. *CoRR*, abs/1811.06128, 2018.

Matthieu Courbariaux and Yoshua Bengio. BinaryNet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

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

Bjarne Grimstad and Henrik Andersson. ReLU networks as surrogate models in mixed-integer linear programs. *Computers & Chemical Engineering*, 131:106580, 2019.

Gurobi Optimization. Gurobi optimizer reference manual, 2019. http://www.gurobi.com.

Junhao Huang, Weize Sun, and Lei Huang. Deep neural networks compression learning based on multiobjective evolutionary algorithms. *Neurocomputing*, 378:260–269, 2020.

Rodrigo Toro Icarte, León Illanes, Margarita P. Castro, André A. Ciré, Sheila A. McIlraith, and J. Christopher Beck. Training binarized neural networks using MIP and CP. In *Proceedings of CP'19*, volume 11802 of *Lecture Notes in Computer Science*, pages 401–417. Springer, 2019.

Ronny Kohavi and Barry Becker. UCI machine learning repository, 1996. http://archive.ics.uci.edu/ml/datasets/adult.

Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *Proceedings of ICLR'19*. OpenReview.net, 2019.