NEUR2RO: NEURAL TWO-STAGE ROBUST OPTIMIZATION

Justin Dumouchelle* University of Toronto
Esther Julien TU Delft
Jannis Kurtz University of Amsterdam
Elias B. Khalil University of Toronto

ABSTRACT

Robust optimization provides a mathematical framework for modeling and solving decision-making problems under worst-case uncertainty. This work addresses two-stage robust optimization (2RO) problems (also called *adjustable robust optimization*), wherein first-stage and second-stage decisions are made before and after uncertainty is realized, respectively. This results in a nested min-max-min optimization problem which is extremely challenging computationally, especially when the decisions are discrete. We propose `Neur2RO`, an efficient machine learning-driven instantiation of column-and-constraint generation (CCG), a classical iterative algorithm for 2RO. Specifically, we learn to estimate the value function of the second-stage problem via a novel neural network architecture that is easy to optimize over by design. Embedding our neural network into CCG yields high-quality solutions quickly as evidenced by experiments on two 2RO benchmarks, knapsack and capital budgeting. For knapsack, `Neur2RO` finds solutions that are within roughly 2% of the best-known values in a few seconds compared to the three hours of the state-of-the-art exact branch-and-price algorithm; for larger and more complex instances, `Neur2RO` finds even better solutions. For capital budgeting, `Neur2RO` outperforms three variants of the k-adaptability algorithm, particularly on the largest instances, with a 10 to 100-fold reduction in solution time. Our code and data are available at `https://github.com/khalil-research/Neur2RO`.

1 INTRODUCTION

A wide range of real-world optimization problems in logistics, finance, and healthcare, among others, can be modeled by discrete optimization models (Petropoulos et al., 2023). While such mixed-integer (linear) problems (MILP) can still be challenging to solve, the problem size that can be tackled with modern solvers has increased significantly thanks to algorithmic developments (Wolsey, 2020; Achterberg & Wunderling, 2013). In recent years, the incorporation of Machine Learning (ML) models into established algorithmic frameworks has received increasing attention (Zhang et al., 2023; Bengio et al., 2021).

While most of ML for discrete optimization has focused on deterministic problems, in many cases, decision-makers face uncertainty in the problem parameters, e.g., due to forecasting or measurement errors in quantities of interest such as customer demand in inventory management. Besides the stochastic optimization approach, for which learning-based heuristics have been proposed recently (Dumouchelle et al., 2022), another popular approach to incorporate uncertainty into optimization models is *robust optimization*, where the goal is to find solutions which are optimal considering the worst realization of the uncertain parameters in a pre-defined uncertainty set (Ben-Tal et al., 2009). This more conservative approach has been extended to two-stage robust problems (2RO) where some of the decisions can be made on the fly after the uncertain parameters are realized (Ben-Tal et al., 2004); see Yanıkoğlu et al. (2019) for a survey.

Example (Capital Budgeting). As a classical example of a two-stage robust problem, consider the capital budgeting problem as defined in Subramanyam et al. (2020) where a company decides to invest in a subset of n projects. Each project i has an uncertain cost $c_i(\boldsymbol{\xi})$ and an uncertain profit

*Corresponding author: justin.dumouchelle@mail.utoronto.ca

$r_i(\boldsymbol{\xi})$ that both depend on the nominal cost and profit, respectively, and some risk factor $\boldsymbol{\xi}$ that dictates the difference from the nominal values to the actual ones. This risk factor, which we call an uncertain scenario, is contained in a given uncertainty set Ξ. The company can invest in a project either before or after observing the risk factor $\boldsymbol{\xi}$, up to a budget B. In the latter case, the company generates only a fraction η of the profit, which reflects a penalty of postponement. The objective of the capital budgeting problem is to maximize the total revenue subject to the budget constraint. This problem can be formulated as:

$$\max_{\mathbf{x}\in\mathcal{X}} \min_{\boldsymbol{\xi}\in\Xi} \max_{\mathbf{y}\in\mathcal{Y}} \quad \mathbf{r}(\boldsymbol{\xi})^\intercal(\mathbf{x}+\eta\mathbf{y}) \tag{1a}$$

$$\text{s.t.} \quad \mathbf{x}+\mathbf{y} \leq \mathbf{1} \tag{1b}$$

$$\mathbf{c}(\boldsymbol{\xi})^\intercal(\mathbf{x}+\mathbf{y}) \leq B. \tag{1c}$$

Here, $\mathcal{X} = \mathcal{Y} = \{0,1\}^n$ and x_i and y_i are the binary variables that indicate whether the company invests in the i-th project in the first- or second-stage, respectively. Constraint 1b ensures that the company can invest in each project only once and constraint 1c ensures that the total cost does not exceed the budget.

2RO with integer decisions is much harder to solve than deterministic MILPs, especially when the uncertain parameters appear in the constraints and the second-stage decisions are discrete. Even evaluating the objective value of a solution in this case is algorithmically challenging (Zhao & Zeng, 2012). In Subramanyam et al. (2020), none of the generated capital budgeting instances could be solved even approximately in a two-hour time limit for $n = 25$, terminating with an optimality gap of around 6%. In contrast to deterministic optimization problems, there is only limited literature on using ML methods to improve robust optimization (Julien et al., 2022; Goerigk & Kurtz, 2022).

Contributions. We propose Neural Two-stage Robust Optimization (`Neur2RO`), an ML framework that can quickly compute high-quality solutions for 2RO. Our contributions are as follows:

- **ML in a novel optimization setting:** 2RO (also known as *adjustable RO*) has been receiving increased interest from the operations research community (Yanıkoğlu et al., 2019) and our work is one of the first to leverage ML in this setting.
- **ML at the service of a classical optimization algorithm:** to deal with the highly constrained nature of real-world optimization problems and rather than attempting to predict solutions directly, we "neuralize" a well-established 2RO algorithm, a strategy that combines the best of both worlds: correctness of an established algorithm with the predictive capabilities of an accurate neural network.
- **A compact, generalizable neural architecture** that is MILP-representable and estimates the thorny component of a 2RO problem, namely the value of the second-stage problem. The network is invariant to problem size and parameters, allowing, for example, the use of the same architecture for capital budgeting instances with a different number of projects and budget parameters.
- **Competitive experimental results** on capital budgeting and a two-stage robust knapsack problem, both benchmarks in the 2RO literature. `Neur2RO` finds solutions that are of similar quality to or better than the state of the art. Large instances benefit the most from our method, with $100\times$ reduction and 10 to $100\times$ reductions in running time for knapsack and capital budgeting, respectively.

2 Background

2.1 Two-Stage Robust Optimization

2RO problems involve two types of decisions. The first set of decisions, $\mathbf{x}$, are referred to as *here-and-now* decisions and are made before the uncertainty is realized. The second set of decisions, $\mathbf{y}$, are referred to as the *wait-and-see* decisions and can be made on the fly after the uncertainty is realized. The uncertain parameters $\boldsymbol{\xi}$ are assumed to be contained in a convex and bounded uncertainty set $\Xi \subset \mathbb{R}^q$. The 2RO problem aims at finding a first-stage solution $\mathbf{x}$ which minimizes the worst-case objective value over all scenarios $\boldsymbol{\xi} \in \Xi$, where for each scenario the best possible

second-stage decision $\mathbf{y}(\boldsymbol{\xi})$ is implemented. Mathematically, a 2RO problem is given by

$$\min_{\mathbf{x}\in\mathcal{X}} \max_{\boldsymbol{\xi}\in\Xi} \min_{\mathbf{y}\in\mathcal{Y}} \quad \mathbf{c}(\boldsymbol{\xi})^\intercal \mathbf{x} + \mathbf{d}(\boldsymbol{\xi})^\intercal \mathbf{y} \tag{2a}$$

$$\text{s.t.} \quad T(\boldsymbol{\xi})\mathbf{x} + W(\boldsymbol{\xi})\mathbf{y} \leq \mathbf{h}(\boldsymbol{\xi}), \tag{2b}$$

where $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{Y} \subseteq \mathbb{R}^m$ are feasible sets for the first and second stage decisions, respectively. In this work, we consider the challenging case of integer sets $\mathcal{X}$ and $\mathcal{Y}$. All parameters of the problem, namely $\mathbf{c}(\boldsymbol{\xi}) \in \mathbb{R}^n, \mathbf{d}(\boldsymbol{\xi}) \in \mathbb{R}^m, W(\boldsymbol{\xi}) \in \mathbb{R}^{r\times m}, T(\boldsymbol{\xi}) \in \mathbb{R}^{r\times n}$, and $\mathbf{h}(\boldsymbol{\xi}) \in \mathbb{R}^r$ depend on the scenario $\boldsymbol{\xi}$. We make the following assumption which is satisfied for the capital budgeting problem (and implicitly knapsack, which does not involve constraint uncertainty).

Assumption. For every $\mathbf{x} \in \mathcal{X}$, we have a method which calculates a scenario $\boldsymbol{\xi} \in \Xi$ for which the second-stage constraints $T(\boldsymbol{\xi})\mathbf{x} + W(\boldsymbol{\xi})\mathbf{y} \leq \mathbf{h}(\boldsymbol{\xi})$ over $\mathbf{y} \in \mathcal{Y}$ are infeasible or verifies that no such scenario exists.

Both single- and multi-stage robust mixed integer problems are NP-hard even for deterministic problems which can be solved in polynomial time Buchheim & Kurtz (2018). Compared to single-stage problems, which are often computationally tractable as they can be solved using reformulations Ben-Tal et al. (2009) or constraint generation Mutapcic & Boyd (2009), two-stage problems are much harder to solve. When dealing with integer first-stage and continuous recourse, CCG is one of the key approaches Zeng & Zhao (2013); Tsang et al. (2023). However, many problems, such as the ones we study here, deal with (mixed-)integer second-stage decisions. While an extension of CCG has been proposed that is able to handle mixed-integer recourse Zhao & Zeng (2012), this method is not well-established and often intractable and the results do not apply for pure integer second-stage problems.

In the case that the uncertainty only appears in the objective function, the 2RO can be solved by oracle-based branch-and-bound methods (Kämmerling & Kurtz, 2020), branch-and-price (Arslan & Detienne, 2022), or iterative cut generation using Fenchel cuts (Detienne et al., 2024). For special problem structures and binary uncertainty sets, a Lagrangian relaxation can be used to transform 2RO problems with constraint uncertainty into 2RO with objective uncertainty which can then be solved by the aforementioned methods (Subramanyam, 2022; Lefebvre et al., 2023).

2.2 Column-and-Constraint Generation

The main idea of CCG is to iterate between a *main problem* (MP) and an *adversarial problem* (AP). The MP is a relaxation of the original problem that only considers a finite subset of the uncertainty set $\Xi' \subset \Xi$. The latter problem can be modeled as a MILP by introducing copies of the second-stage decision variables $\mathbf{y}$ for each of the scenarios in Ξ'. After calculating an optimal solution of the MP, the AP finds new scenarios in the uncertainty set that cut off the current solution in the MP. When no such scenario can be found, the optimality of the current MP solution is guaranteed. For mathematical formulations of the two problems and a more detailed description of the CCG procedure, see Figure 1 and Appendix B.1.

CCG often fails to calculate an optimal solution in reasonable time since both the MP and the AP are very hard to solve. In each iteration, the size of the MP increases leading to it being difficult to solve to optimality even with commercial MILP solvers such as Gurobi (Gurobi Optimization, LLC, 2023). Furthermore, solving the AP is extremely challenging for integer second-stage variables. In Zhao & Zeng (2012), the authors present a column-and-constraint algorithm which solves the AP if the second stage is a mixed-integer problem; this leads to a CCG for the AP inside the main CCG, a most intractable combination. Additionally, the method of Zhao & Zeng (2012) is not applicable to purely integer second-stage decisions such as the problems we consider here.

3 Methodology

At a high level, our approach aims to train a neural network that predicts the optimal second-stage objective value function and then integrates this model within a CCG framework to obtain first-stage decisions. We rely on a training dataset of historical instances that can be used or generated, as is typically assumed in ML-for-optimization work.

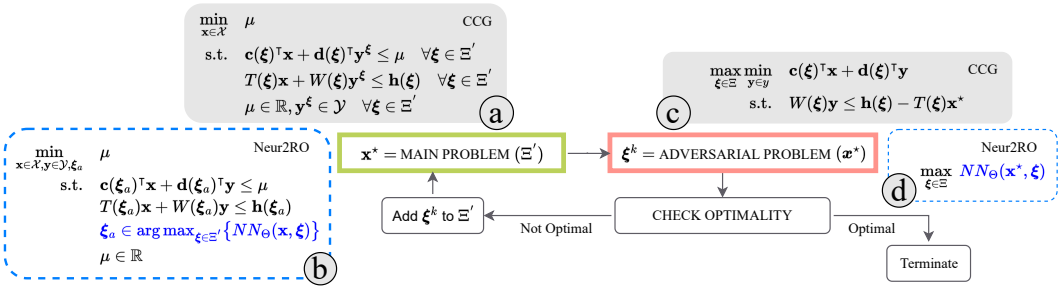

Figure 1: Column-and-constraint generation: in each iteration, a *main problem* (box (a)) is solved to find a good first-stage solution $\mathbf{x}^\star$ for the set of scenarios that have been identified thus far (initially, none). Then, an *adversarial problem* (box (c)) is solved to obtain a new scenario for which the solution $\mathbf{x}^\star$ is not feasible anymore in MP. If no such scenario exists, then $\mathbf{x}^\star$ is *optimal* and CCG terminates. Otherwise, the adversarial scenario is added to the set of worst-case scenarios and we iterate to MP. For each of the MP and AP, we show two versions: classical (CCG, boxes (a) and (c)) and learning-augmented (`Neur2RO`, dashed boxes (b) and (d)).

3.1 Learning Model

As mentioned before, CCG is computationally very expensive. Both the MP and AP contribute to its intractability (see Figure 1 boxes (a) and (c) for descriptions). In the MP, for each added scenario, a new second-stage decision $\mathbf{y}$ is introduced. When a large number of scenarios are required to obtain a robust solution, the number of variables grows rapidly. Moreover, the AP is especially hard when the second-stage decisions are integer, which is the case we consider. In our learning-augmented approach, we replace the intractable elements of the CCG with MILP representations of a trained NN which is computationally much easier to handle (Figure 1).

We train a neural network that can accurately predict the optimal value of the second-stage problem for a given input of a first-stage decision, an uncertainty realization, and the problem's specification, $\mathcal{P}$. The problem specification refers to the coefficients and size of the optimization problem, e.g., the nominal values of the profit and costs in the capital budgeting problem. More formally we train a neural network $NN_\Theta(\cdot)$, to approximate the optimal value of the integer problem

$$NN_\Theta(\mathbf{x}, \boldsymbol{\xi}, \mathcal{P}) \approx \min_{y \in \mathcal{Y}} \{\mathbf{c}_\mathcal{P}(\boldsymbol{\xi})^\intercal \mathbf{x} + \mathbf{d}_\mathcal{P}(\boldsymbol{\xi})^\intercal \mathbf{y} : W_\mathcal{P}(\boldsymbol{\xi})\mathbf{y} \leq \mathbf{h}_\mathcal{P}(\boldsymbol{\xi}) - T_\mathcal{P}(\boldsymbol{\xi})\mathbf{x}\}, \tag{3}$$

where Θ are the weights of the neural network. For ease of notation, we hereafter omit $\mathcal{P}$ in the formulation. For more details on the architecture of $NN_\Theta(\cdot)$, see Section 3.3. Alternatively, as $\mathbf{c}_\mathcal{P}(\boldsymbol{\xi})^\intercal \mathbf{x}$ is a scalar product of the input vectors, we instead could only predict the second-stage objective, i.e., $\mathbf{d}_\mathcal{P}(\boldsymbol{\xi})^\intercal \mathbf{y}$, subject to the same constraints. However, as demonstrated in Appendix H.3, predicting the sum of first- and second-stage objectives achieves higher-quality solutions.

3.2 ML-Based Column-and-Constraint Generation

Having defined the learning task, we now describe the ML-based approximate CCG algorithm. For an overview of this method, see Figure 1.

Main Problem. Given a finite subset of scenarios $\Xi' \subset \Xi$, we reformulate the MP using an $\arg\max$ operator which selects a scenario that achieves the worst objective function value when replacing the second-stage objective value by the neural network formulation.

$$\min_{\mathbf{x} \in \mathcal{X}, \mathbf{y} \in \mathcal{Y}, \boldsymbol{\xi}_a \in \Xi} \quad \mathbf{c}(\boldsymbol{\xi}_a)^\intercal \mathbf{x} + \mathbf{d}(\boldsymbol{\xi}_a)^\intercal \mathbf{y} \tag{4a}$$

$$\text{s.t.} \quad W(\boldsymbol{\xi}_a)\mathbf{y} + T(\boldsymbol{\xi}_a)\mathbf{x} \leq \mathbf{h}(\boldsymbol{\xi}_a), \tag{4b}$$

$$\boldsymbol{\xi}_a \in \arg\max_{\boldsymbol{\xi} \in \Xi'} \{NN_\Theta(\mathbf{x}, \boldsymbol{\xi})\}. \tag{4c}$$

Modeling the $\arg\max$ can be done by adding additional linear constraints and binary variables, which we explicitly show in Appendix C. The MP results in a MILP formulation.

This formulation is indeed not the only option; for example, one could instead consider another formulation, called $\max$, which provides a more intuitive formulation and does not require modeling second-stage variables; details are provided in Appendix H.1. However, equation 4 has one key property that motivates its efficacy. From the machine learning perspective, rather than requiring a neural network to be an accurate estimator of the objective for each scenario, we only require that the neural network be able to identify the maximal scenario. Prediction inaccuracy is then compensated for in equation 4a by exactly modeling the second-stage cost. As a result, when solving the MP, the true optimal first-stage decision for the selected scenario will be the minimizer, rather than a potentially suboptimal first-stage decision based on any inaccuracy of the learning model. Appendix H.1 presents an ablation comparing both the solution quality of the $\arg\max$ and $\max$ formulations on the knapsack problem, and establishes that the $\arg\max$ formulation indeed computes higher quality solutions across every instance.

Adversarial Problem. In the AP of `Neur2RO`, we replace the inner optimization problem over $\mathbf{y}$ by its prediction $NN_\Theta(\mathbf{x}^\star, \boldsymbol{\xi})$, where $\mathbf{x}^\star$ is given (see Figure 1 box (d)). When we deal with constraint uncertainty, we first check if there exists a scenario $\boldsymbol{\xi} \in \Xi$, such that no feasible $\mathbf{y} \in \mathcal{Y}$ exists for the constraints

$$W(\boldsymbol{\xi})\mathbf{y} + T(\boldsymbol{\xi})\mathbf{x}^\star \leq \mathbf{h}(\boldsymbol{\xi}),$$

which we can do by the assumption from Section 2.1. If such a scenario exists, we add it to Ξ' and continue with solving the MP again. Note that in this case $\mathbf{x}^\star$ is not feasible for MP in the next iteration. If no such scenario could be found, we calculate an optimal solution $\boldsymbol{\xi}^\star$ of the AP in box (d) of Figure 1 which can be done by using the MILP representation for NN. Note that if Ξ is a polyhedron or an ellipsoid, then this problem results in a mixed-integer linear or quadratic problem, respectively, which can be solved by state-of-the-art solvers such as Gurobi. We compare the optimal value of the latter problem with the objective values of all scenarios that were considered in the MP before. If the following holds

$$\max_{\boldsymbol{\xi}\in\Xi} NN_\Theta(\mathbf{x}^\star, \boldsymbol{\xi}) \geq \max_{\boldsymbol{\xi}\in\Xi'} NN_\Theta(\mathbf{x}^\star, \boldsymbol{\xi}) + \varepsilon \tag{5}$$

for a pre-defined accuracy parameter $\varepsilon > 0$, then we add $\boldsymbol{\xi}^\star$ to Ξ' and continue with the MP. Otherwise, we stop the algorithm. Finally, note that we can calculate both types of scenarios in each iteration and add them both to Ξ' before we iterate to the MP. As the adversarial problem requires finding the worst-case uncertainty over a neural network's input, heuristic approaches may significantly improve solving time with minimal degradation in solution quality. Appendix H.2 presents an ablation demonstrating significantly lower solution time at a minimal cost of solution quality for sampling- and relaxation-based heuristics.

Convergence. Since our algorithm does not apply the standard CCG steps, the convergence guarantee from the classical algorithm does not hold. However, we prove in Appendix F that it holds if only finitely many first-stage solutions exist, which is the case if all first-stage variables are integer and $\mathcal{X}$ is bounded; this indeed holds for the knapsack and capital budgeting problems.

Theorem 1. *If $\mathcal{X}$ is finite, the ML-based CCG terminates after a finite number of iterations.*

3.3 Architecture

For the ML-based CCG, one requirement is the optimization, in each iteration, over several trained neural networks in the MP (one for each scenario in equation 4c) and a single trained neural network in the AP. Generally, increasing the size of the networks will lead to more challenging and potentially intractable optimization problems. For that reason, developing an architecture that can be efficiently optimized over is a crucial aspect of an efficient ML-based CCG algorithm.

To achieve efficient optimization, we embed the first-stage decisions, $\mathbf{x}$, and a scenario, $\boldsymbol{\xi}$, into low-dimensional embeddings using networks $\Phi_\mathbf{x}$ and $\Phi_{\boldsymbol{\xi}}$, respectively. These embeddings are concatenated and passed through a final small neural network (the *Value Network*) Φ that predicts the objective of the optimal second-stage response; see Figure 2 for a pictorial representation.

Main Problem Optimization. When representing the trained models in the MILP, we only have to represent the embedding network $\Phi_\mathbf{x}$ and the small value network Φ, which can be done by classical MILP representations of ReLU NNs (Fischetti & Jo, 2018). Since the scenario parameters are not variables here, the scenario embeddings $\Phi_{\boldsymbol{\xi}}(\boldsymbol{\xi}^k)$ can be precomputed via a forward pass

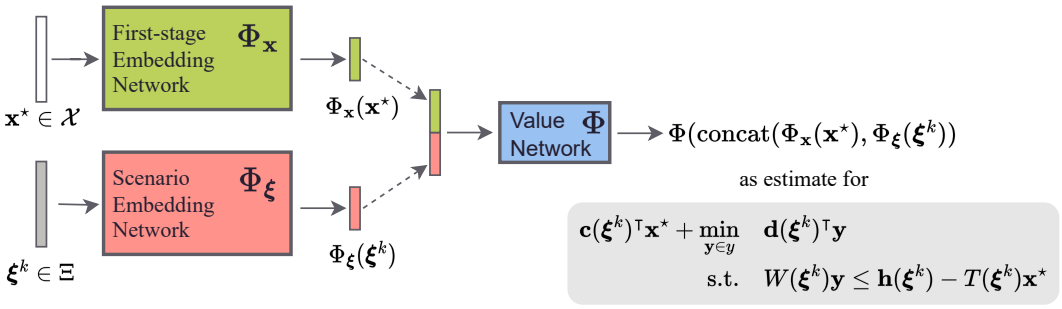

Figure 2: Neural network architecture for ML-based CCG. The current first-stage solution, $\mathbf{x}^\star$, is embedded once using the network $\Phi_{\mathbf{x}}(\cdot)$. A scenario $\boldsymbol{\xi}^k$ is embedded using the network $\Phi_{\boldsymbol{\xi}}(\cdot)$. To estimate the value of the second-stage optimization problem corresponding to a particular pair $(\mathbf{x}^\star, \boldsymbol{\xi}^k)$, the two embedding vectors are concatenated into one (dashed arrows) and then passed into the final *Value Network*.

for each scenario, i.e., no MILP representation is needed for $\Phi_{\boldsymbol{\xi}}$. If Φ is a small neural network, then representing a large number of copies of the network (one per scenario) remains amenable to efficient optimization.

Adversarial Problem Optimization. For the AP, we only require representing $\Phi_{\boldsymbol{\xi}}$ and Φ as the embedding of $\mathbf{x}$ can be precomputed with a forward pass.

Generalizing Across Instances. For simplicity of notation and presentation, the previous sections have omitted the generalization across instances, which is a key aspect of the generality of our methodology. To generalize across instances, invariance to the number, ordering, constraint coefficients, and objective coefficients of decision variables is required. To handle this, `Neur2RO` leverages set-based neural networks (Zaheer et al., 2017) for $\Phi_{\mathbf{x}}$ and $\Phi_{\boldsymbol{\xi}}$. Specifically, embeddings are computed for each single first-stage and scenario variable (x_i and ξ_i) using their values, constraints, and objective coefficients, via a network with shared parameters. These embeddings are then aggregated and passed through an additional feed-forward neural network to derive the first-stage and scenario embeddings. For a detailed diagram of this architecture, see Appendix D.

4 Experimental Setup

Computational Setup. All experiments were run on a computing cluster with an Intel Xeon CPU E5-2683 and Nvidia Tesla P100 GPU with 64GB of RAM (for training). Pytorch 1.12.1 (Paszke et al., 2019) was used for all learning models. Gurobi 10.0.2 (Gurobi Optimization, LLC, 2023) was used as the MILP solver and gurobi-machinelearning 1.3.0 was used to embed the neural networks into MILPs. For evaluation, all solving was limited to 3 hours. For `Neur2RO`, we terminate a solve of the MP or AP early if no improvement in solution is observed in 180 seconds. Our code and data are at `https://github.com/khalil-research/Neur2RO`.

2RO Problems. We benchmark `Neur2RO` on two 2RO problems from the literature, namely a two-stage knapsack problem and the capital budgeting problem. In both cases, our instances are as large or larger than considered in the literature. The two-stage knapsack problem is in the first stage a classical knapsack problem. The second stage has decisions for responding to an uncertain profit degradation. The capital budgeting problem is described in the introduction. For a detailed description of these problems, see Appendix A. Below we briefly detail each problem.

- **Knapsack.** For the knapsack problem, we use the same instances as in Arslan & Detienne (2022), which have been inspired by Ben-Tal et al. (2009). They have categorized their instances into four groups: uncorrelated (UN), weakly correlated (WC), almost strongly correlated (ASC), and strongly correlated (SC), which affects the correlation of the nominal profits of items with their cost and, in turn, the difficulty of the problem. More correlated instances are much harder to solve. We consider instances of sizes $n \in \{20, 30, 40, 50, 60, 70, 80\}$.

- **Capital budgeting.** These problem instances are generated similar to Subramanyam et al. (2020).While uncertain parameters appear in the constraints (see equation 1c), we can easily verify the assumption given in Section 2.1 as follows: for every $\mathbf{x}$ we check if $\max_{\boldsymbol{\xi} \in \Xi} \mathbf{c}(\boldsymbol{\xi}) \leq B$, where the maximum can be easily calculated since it is a linear problem over Ξ. If the latter inequality is true, the second-stage problem is feasible since we can choose $\mathbf{y} = \mathbf{0}$. On the other hand, if the inequality is violated, then no feasible second-stage solution exists since $\mathbf{y} \geq \mathbf{0}$, and hence the maximizing scenario will be added to MP. We consider instances of sizes $n \in \{10, 20, 30, 40, 50\}$.

Baselines. For knapsack, we compare to the branch-and-price (BP) algorithm from Arslan & Detienne (2022), the state-of-the-art for 2RO problems with objective uncertainty. We use the instances and the objective values and solution times reported in their paper (link). For capital budgeting, we use the k-adaptability approach of Subramanyam et al. (2020) (more details in Appendix B.2) with $k = 2, 5, 10$ as a baseline; CCG is not tractable for this problem due to its integer recourse.

Evaluation. After `Neur2RO` finds the first-stage decision $\mathbf{x}^\star$, we obtain the corresponding objective value by solving 2 for a fixed $\mathbf{x}$. For knapsack, this can be efficiently solved by constraint generation of $\mathbf{y}$. For capital budgeting on the other hand, due to constraint uncertainty, we cannot use the constraint generation approach. Instead we sample scenarios from Ξ, and solve 2 with fixed $\mathbf{x}$ and $\boldsymbol{\xi}$. See Appendix E for a more detailed explanation of these methods.

As previously mentioned, we use k-adaptability as the baseline for the capital budgeting problem. This method solves 2 only approximately with the approximation quality getting better with larger k at an increase in solution times. We take the first-stage solution found by k-adaptability and compare it with the one of `Neur2RO` using the scenario sampling approach just described.

For the evaluation of the objective values, two metrics are considered. We use relative error (RE), i.e., the gap to the best-known solution, to compare solution quality. Specifically, if $obj^\star$ is the value of the best solution found by `Neur2RO` or a baseline for a particular instance, then for algorithm A with objective obj_A, the RE is given by $100 \cdot \frac{|obj^\star - obj_A|}{|obj^\star|}$. To compare efficiency, we compare the average solution time.

Data Collection & Training. For data collection, we sample sets of instances, first-stage decisions, and scenarios to obtain features. The features are provided in Appendix I. Labels are then computed by solving the corresponding innermost optimization problem, i.e., a tractable deterministic MILP as both $\mathbf{x}$ and $\boldsymbol{\xi}$ are fixed. Additionally, this process is highly parallelizable since each optimization problem is independent. For knapsack and capital budgeting, we randomly sample 500 instances, 10 first-stage decisions per instance, and 50 scenarios per first-stage decision, resulting in 250,000 data points. The dataset is split into 200,000 and 50,000 samples for training and validation, respectively.

We train one size-independent model for each problem for 500 epochs. The data collection times, training times, and total times (in seconds) are 2,162, 3,789, and 5,951 for knapsack and 3,212, 2,195, and 5,407 for capital budgeting. We note that both times are relatively insignificant given that we provide approximately twice the time (3 hours) to solve a single instance during evaluation. Furthermore, the model for `Neur2RO` generalizes across instance parameters and sizes. Appendix I provides full detail on model hyperparameters and training.

5 Experimental Results

For knapsack, we test our method and the baseline on 18 instances per correlation type and instance size (504 instances). For capital budgeting, we test on 50 instances per instance size (250 instances). We note that training and validation data are generated using the procedures specified in the corresponding papers, and different instances are used for testing. For optimization of `Neur2RO`, this section presents results for solving the MIP and MILP formulations for the MP and AP, with the $\arg\max$ formulation outlined in Section 3 for the MP. Tables 1-2 report the median RE and solving times. In addition, for more detailed distributional information, boxplots and more detailed metrics, are provided in Appendix G.

Knapsack. Table 1 demonstrates a clear improvement in scalability, with the solving time of `Neur2RO` ranging between 4 and 77 seconds, while the solving time for BP scales directly with difficulty induced by the size and correlation type. For the more difficult instances, i.e., instances

Correlation Type	# items	Median RE		Times	
		Neur2RO	BP	Neur2RO	BP
Uncorrelated	20	1.417	**0.000**	4	**0**
	30	1.188	**0.000**	6	**1**
	40	1.614	**0.000**	9	**3**
	50	1.814	**0.000**	**9**	12
	60	1.146	**0.000**	**14**	18
	70	1.408	**0.000**	**16**	46
	80	0.968	**0.000**	**11**	388
Weakly Correlated	20	1.582	**0.000**	**5**	29
	30	2.236	**0.000**	**11**	454
	40	1.595	**0.000**	**20**	6179
	50	1.757	**0.000**	**19**	8465
	60	0.695	**0.000**	**77**	9242
	70	0.165	**0.000**	**15**	10800
	80	**0.000**	0.341	**21**	10800
Almost Strongly Correlated	20	1.439	**0.000**	**5**	9
	30	0.782	**0.000**	**6**	2708
	40	0.497	**0.000**	**10**	4744
	50	0.019	**0.000**	**7**	8852
	60	**0.000**	0.016	**14**	10261
	70	**0.017**	0.031	**13**	10800
	80	**0.000**	0.265	**12**	10800
Strongly Correlated	20	1.604	**0.000**	**5**	9
	30	0.610	**0.000**	**7**	2473
	40	0.443	**0.000**	**11**	5665
	50	0.073	**0.000**	**9**	8240
	60	0.042	**0.010**	**11**	10800
	70	**0.020**	0.027	**16**	10800
	80	**0.000**	0.179	**13**	10800

Table 1: Median RE and solving times for knapsack instances. For each row, the median RE and average solving time are computed over 18 instances. All times in seconds. The smallest (best) values in each row/metric are in bold.

# items	Median RE				Times			
	Neur2RO	$k=2$	$k=5$	$k=10$	Neur2RO	$k=2$	$k=5$	$k=10$
10	1.105	1.140	**0.000**	**0.000**	59	**20**	9561	10800
20	**0.000**	0.196	0.112	0.064	**324**	8702	10800	10800
30	0.109	**0.020**	0.073	0.032	**602**	10801	10800	10800
40	**0.009**	0.074	0.011	0.019	**739**	10806	10801	10801
50	**0.001**	0.033	0.039	0.020	**1032**	10807	10804	10801

Table 2: Combined results for capital budgeting instances. For each row, the median RE and average solving time are computed over 50 instances. All times in seconds. The smallest (best) values in each row/metric are in bold.

with a large number of items and (almost) strong correlation, Neur2RO generally finds better quality solutions over 100 times faster than BP, which is a very strong result considering BP is the state-of-the-art for problems with objective uncertainty. Figures 6-7 of Appendix G further demonstrate that the distribution of RE achieved by Neur2RO, not just the median, is far more favorable than BP's on the most challenging instances. For easier instances, Neur2RO is less competitive in terms of solution quality as BP converges to optimal solutions within the time limit. However, even for these instances, Neur2RO achieves a median RE of 2.235% in the worst-case, often still 1-2 orders of magnitude faster than BP, with the exception of a few very easy instances.

Capital budgeting. Neur2RO achieves the lowest median RE for 20, 40, and 50-item instances, i.e., the two largest and most challenging instance sets. The distribution of RE for 40 and 50-item instances provided in Figure 8 of Appendix G is indeed consistent with the median result, as it illustrates that Neur2RO finds quality solutions on the majority of the instances. In terms of solving time, Neur2RO generally converges much faster than k-adaptability, resulting in a very favorable trade-off: we can find better or equally good solutions 10 to 100 times faster. Note that the relative errors are quite small in an absolute sense. For example, for 30-item instances, Neur2RO has a median RE of 0.109 compared to the best baseline's 0.020; solutions that are within 0.109% of the best achievable may be acceptable in practice. Note that we have also measured the median RE for k-adaptability assuming a shorter time limit, namely the same amount of time as Neur2RO on each instance. Taking the incumbent solution found by k-adaptability at that time point typically yields worse solutions than those reported in Table 2, see Appendix H.4 for details. Compared to the knapsack, the solving time is generally much larger as the instance size increases. We speculate that this may relate to the uncertainty in the objective of the first-stage decision or the budget constraints that are not present in the knapsack problem.

In summary, for both benchmark problems, Neur2RO achieves high-quality solutions. For relatively easy or small instances, state-of-the-art methods sometimes find slightly better solutions, often at a much higher computational cost. However, as the instances become more difficult, Neur2RO demonstrates a clear improvement in overall solution quality and computing time.

6 Related Work

Robust optimization. Besides the exact solution methods mentioned in Section 2.1, several heuristic methods have been developed to derive near-optimal solutions for mixed-integer 2RO problems. Methods that solve 2RO heuristically are k-adaptability (Bertsimas & Caramanis, 2010; Hanasusanto et al., 2015; Subramanyam et al., 2020), decision rules (Bertsimas & Georghiou, 2018; 2015), and iteratively splitting the uncertainty set (Postek & Hertog, 2016). Machine Learning techniques have been developed to speed up solution algorithms for the k-adaptability problem in Julien et al. (2022). In Goerigk & Kurtz (2022) a decision tree classifier is trained to predict good start scenarios for the CCG. While being heuristic solvers, all of the above methods are still computationally highly demanding. In this paper, the k-adaptability branch-and-bound algorithm by Subramanyam et al. (2020) is used as a baseline since it is one of the only methods which is able to calculate high quality solutions for reasonable problem sizes. For an elaborate overview of the latter algorithm, see Appendix B.2.

Besides improving algorithmic performance, ML methods have been used to construct uncertainty sets based on historical data. In Goerigk & Kurtz (2023) one-class neural networks are used to construct highly complex and non-convex uncertainty sets. Results from statistical learning theory are used to derive guarantees for ML designed uncertainty sets in Tulabandhula & Rudin (2014). Other approaches use principal component analysis and kernel smoothing (Ning & You, 2018), support vector clustering (Shang et al., 2017; Shang & You, 2019; Shen et al., 2020), statistical hypothesis testing (Bertsimas et al., 2018), or Dirichlet process mixture models (Ning & You, 2017; Campbell & How, 2015). In Wang et al. (2023a) uncertainty sets providing a certain probabilistic guarantee are derived by solving a CVaR-constrained bilevel problem by an augmented Lagrangian method. While interesting and related, we here assume the uncertainty set is known.

MILP representations of neural networks. One key aspect of `Neur2RO` is representing neural networks as constraints and variables in MILPs, which was first explored in Cheng et al. (2017); Tjeng et al. (2017); Fischetti & Jo (2018); Serra et al. (2018). These representations have motivated active research to improve the MILP solving efficiency of optimizing over-trained models Grimstad & Andersson (2019); Anderson et al. (2020); Wang et al. (2023b), as well as several software contributions Bergman et al. (2022); Ceccon et al. (2022), in addition to Gurobi, a commercial MILP solver, providing an open-source library. The use of embedding trained predictive models to derive approximate MILPs has been explored for non-linear constraints or intractable constraints (Say et al., 2017; Grimstad & Andersson, 2019; Murzakhanov et al., 2020; Katz et al., 2020; Kody et al., 2022), stochastic programming (Dumouchelle et al., 2022; Kronqvist et al., 2023), and bilevel optimization (Dumouchelle et al., 2024). As `Neur2RO` is based on an approximation for intractable 2RO problems with embedded neural networks, the latter area of research is the most closely related. However, the min-max-min optimization in 2RO renders previous learning-based MILP approximations unsuitable.

7 Conclusion

With the uncertainty in real-world noisy data, the economy, the climate, and other avenues, there is an increasing need for efficient robust decision-making. We have shown how `Neur2RO` uses MILP-representable feedforward neural networks to estimate the thorny component of a family of two-stage robust optimization instances, namely the value of the second-stage problem. The neural network architecture delicately combines low-dimensional embeddings of a first-stage decision and a scenario to produce the second-stage value estimate. Using an off-the-shelf MILP solver, we then use the neural network in a classical iterative algorithm for 2RO. `Neur2RO` to find competitive solutions compared to state-of-the-art methods on two challenging benchmark problems, knapsack and capital budgeting, at a substantial reduction in solution time.

Our work paves the way for further integration of learning and optimization under uncertainty. The AP in the CCG algorithm is over the inputs of a neural network, which could benefit from the many "adversarial attack" heuristics in the ML literature. Generalizations of `Neur2RO` that make predictions of the infeasibility of a particular constraint could be explored. ReLU networks are not the only class of models that is MILP-representable; decision tree models could be used as alternatives and may be appropriate for other problem settings.

ACKNOWLEDGEMENTS

Dumouchelle and Khalil acknowledge support from the Scale AI Research Chair Program and an NSERC Discovery Grant. Julien acknowledges funding from the Netherlands Organisation for Scientific Research (NWO).

REFERENCES

Tobias Achterberg and Roland Wunderling. Mixed integer programming: Analyzing 12 years of progress. In *Facets of combinatorial optimization: Festschrift for martin grötschel*, pp. 449–481. Springer, 2013.

Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, pp. 1–37, 2020.

Ayşe N Arslan and Boris Detienne. Decomposition-based approaches for a class of two-stage robust binary optimization problems. *INFORMS journal on computing*, 34(2):857–871, 2022.

Ayşe N Arslan, Michael Poss, and Marco Silva. Min-sup-min robust combinatorial optimization with few recourse solutions. *INFORMS Journal on Computing*, 34(4):2212–2228, 2022.

Aharon Ben-Tal, Alexander Goryashko, Elana Guslitzer, and Arkadi Nemirovski. Adjustable robust solutions of uncertain linear programs. *Mathematical programming*, 99(2):351–376, 2004.

Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust optimization*, volume 28. Princeton university press, 2009.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421, 2021.

David Bergman, Teng Huang, Philip Brooks, Andrea Lodi, and Arvind U Raghunathan. JANOS: an integrated predictive and prescriptive modeling framework. *INFORMS Journal on Computing*, 34(2):807–816, 2022.

Dimitris Bertsimas and Constantine Caramanis. Finite adaptability in multistage linear optimization. *IEEE Transactions on Automatic Control*, 55(12):2751–2766, 2010.

Dimitris Bertsimas and Angelos Georghiou. Design of near optimal decision rules in multistage adaptive mixed-integer optimization. *Operations Research*, 63(3):610–627, 2015.

Dimitris Bertsimas and Angelos Georghiou. Binary decision rules for multistage adaptive mixed-integer optimization. *Mathematical Programming*, 167:395–433, 2018.

Dimitris Bertsimas, Vishal Gupta, and Nathan Kallus. Data-driven robust optimization. *Mathematical Programming*, 167:235–292, 2018.

Christoph Buchheim and Jannis Kurtz. Robust combinatorial optimization under convex and discrete cost uncertainty. *EURO Journal on Computational Optimization*, 6(3):211–238, 2018.

Trevor Campbell and Jonathan P How. Bayesian nonparametric set construction for robust optimization. In *2015 American Control Conference (ACC)*, pp. 4216–4221. IEEE, 2015.

Francesco Ceccon, Jordan Jalving, Joshua Haddad, Alexander Thebelt, Calvin Tsay, Carl D Laird, and Ruth Misener. OMLT: Optimization & machine learning toolkit. *arXiv preprint arXiv:2202.02414*, 2022.

Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pp. 251–268. Springer, 2017.

Boris Detienne, Henri Lefebvre, Enrico Malaguti, and Michele Monaci. Adjustable robust optimization with objective uncertainty. *European Journal of Operational Research*, 312(1):373–384, 2024.

Justin Dumouchelle, Rahul Patel, Elias B Khalil, and Merve Bodur. Neur2SP: Neural two-stage stochastic programming. *Advances in Neural Information Processing Systems*, 35, 2022.

Justin Dumouchelle, Esther Julien, Jannis Kurtz, and Elias B Khalil. Neur2BiLO: Neural bilevel optimization. *arXiv preprint arXiv:2402.02552*, 2024.

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

Alireza Ghahtarani, Ahmed Saif, Alireza Ghasemi, and Erick Delage. A double-oracle, logic-based benders decomposition approach to solve the k-adaptability problem. *Computers & Operations Research*, 155:106243, 2023.

Marc Goerigk and Jannis Kurtz. Data-driven prediction of relevant scenarios for robust optimization. *arXiv e-prints*, pp. arXiv–2203, 2022.

Marc Goerigk and Jannis Kurtz. Data-driven robust optimization using deep neural networks. *Computers & Operations Research*, 151:106087, 2023.

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

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL `https://www.gurobi.com`.

Grani A Hanasusanto, Daniel Kuhn, and Wolfram Wiesemann. K-adaptability in two-stage robust binary programming. *Operations Research*, 63(4):877–891, 2015.

Esther Julien, Krzysztof Postek, and Ş İlker Birbil. Machine learning for k-adaptability in two-stage robust optimization. *arXiv preprint arXiv:2210.11152*, 2022.

Nicolas Kämmerling and Jannis Kurtz. Oracle-based algorithms for binary two-stage robust optimization. *Computational Optimization and Applications*, 77:539–569, 2020.

Justin Katz, Iosif Pappas, Styliani Avraamidou, and Efstratios N. Pistikopoulos. The integration of explicit MPC and ReLU based neural networks. *IFAC-PapersOnLine*, 53(2):11350–11355, 2020. ISSN 2405-8963. doi:https://doi.org/10.1016/j.ifacol.2020.12.544. URL `https://www.sciencedirect.com/science/article/pii/S2405896320308429`. 21st IFAC World Congress.

Alyssa Kody, Samuel Chevalier, Spyros Chatzivasileiadis, and Daniel Molzahn. Modeling the AC power flow equations with optimally compact neural networks: Application to unit commitment. *Electric Power Systems Research*, 213:108282, 2022. ISSN 0378-7796. doi:https://doi.org/10.1016/j.epsr.2022.108282. URL `https://www.sciencedirect.com/science/article/pii/S0378779622004771`.

Jan Kronqvist, Boda Li, Jan Rolfes, and Shudian Zhao. Alternating mixed-integer programming and neural network training for approximating stochastic two-stage problems. *arXiv preprint arXiv:2305.06785*, 2023.

Jannis Kurtz. Approximation algorithms for min-max-min robust optimization and k-adaptability under objective uncertainty. *arXiv preprint arXiv:2106.03107*, 2023.

Henri Lefebvre, Enrico Malaguti, and Michele Monaci. Adjustable robust optimization with discrete uncertainty. *INFORMS Journal on Computing*, 2023.

Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. *Advances in neural information processing systems*, 27, 2014.

Ilgiz Murzakhanov, Andreas Venzke, George S Misyris, and Spyros Chatzivasileiadis. Neural networks for encoding dynamic security-constrained optimal power flow. *arXiv preprint arXiv:2003.07939*, 2020.

Almir Mutapcic and Stephen Boyd. Cutting-set methods for robust convex optimization with pessimizing oracles. *Optimization Methods & Software*, 24(3):381–406, 2009.

Chao Ning and Fengqi You. Data-driven adaptive nested robust optimization: general modeling framework and efficient computational algorithm for decision making under uncertainty. *AIChE Journal*, 63(9):3790–3817, 2017.

Chao Ning and Fengqi You. Data-driven decision making under uncertainty integrating robust optimization with principal component analysis and kernel smoothing methods. *Computers & Chemical Engineering*, 112:190–210, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Fotios Petropoulos, Gilbert Laporte, Emel Aktas, Sibel A Alumur, Claudia Archetti, Hayriye Ayhan, Maria Battarra, Julia A Bennell, Jean-Marie Bourjolly, John E Boylan, et al. Operational research: Methods and applications. *arXiv preprint arXiv:2303.14217*, 2023.

Krzysztof Postek and Dick den Hertog. Multistage adjustable robust mixed-integer optimization via iterative splitting of the uncertainty set. *INFORMS Journal on Computing*, 28(3):553–574, 2016.

Buser Say, Ga Wu, Yu Qing Zhou, and Scott Sanner. Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In *IJCAI*, pp. 750–756, 2017.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. In *International Conference on Machine Learning*, pp. 4558–4566. PMLR, 2018.

Chao Shang and Fengqi You. A data-driven robust optimization approach to scenario-based stochastic model predictive control. *Journal of Process Control*, 75:24–39, 2019.

Chao Shang, Xiaolin Huang, and Fengqi You. Data-driven robust optimization based on kernel learning. *Computers & Chemical Engineering*, 106:464–479, 2017.

Feifei Shen, Liang Zhao, Wenli Du, Weimin Zhong, and Feng Qian. Large-scale industrial energy systems optimization under uncertainty: A data-driven robust optimization approach. *Applied Energy*, 259:114199, 2020.

Anirudh Subramanyam. A lagrangian dual method for two-stage robust optimization with binary uncertainties. *Optimization and Engineering*, 23(4):1831–1871, 2022.

Anirudh Subramanyam, Chrysanthos E Gounaris, and Wolfram Wiesemann. K-adaptability in two-stage mixed-integer robust optimization. *Mathematical Programming Computation*, 12:193–224, 2020.

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

Man Yiu Tsang, Karmel S Shehadeh, and Frank E Curtis. An inexact column-and-constraint generation method to solve two-stage robust optimization problems. *Operations Research Letters*, 51 (1):92–98, 2023.

Theja Tulabandhula and Cynthia Rudin. Robust optimization using machine learning for uncertainty sets. *arXiv preprint arXiv:1407.1097*, 2014.

Irina Wang, Cole Becker, Bart Van Parys, and Bartolomeo Stellato. Learning for robust optimization. *arXiv preprint arXiv:2305.19225*, 2023a.

Keliang Wang, Leonardo Lozano, Carlos Cardonha, and David Bergman. Optimizing over an ensemble of trained neural networks. *INFORMS Journal on Computing*, 35(3):652–674, 2023b.

Laurence A Wolsey. *Integer programming*. John Wiley & Sons, 2020.

İhsan Yanıkoğlu, Bram L Gorissen, and Dick den Hertog. A survey of adjustable robust optimization. *European Journal of Operational Research*, 277(3):799–813, 2019.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Bo Zeng and Long Zhao. Solving two-stage robust optimization problems using a column-and-constraint generation method. *Operations Research Letters*, 41(5):457–461, 2013.

Jiayi Zhang, Chang Liu, Xijun Li, Hui-Ling Zhen, Mingxuan Yuan, Yawen Li, and Junchi Yan. A survey for solving mixed integer programming via machine learning. *Neurocomputing*, 519: 205–217, 2023.

Long Zhao and Bo Zeng. An exact algorithm for two-stage robust optimization with mixed integer recourse problems. *submitted, available on Optimization-Online. org*, 2012.