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 5 to 10-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 $r_i(\boldsymbol{\xi})$ 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. The company can invest in a project either before or after observing the risk factor, up to a budget B. In the latter case, the company generates only a fraction 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:

$$\max_{x \in X} \min_{\xi \in \Xi} \max_{y \in Y} \quad r(\xi)^\top (x + y) \tag{1a}$$

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

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

Here, $X = Y = \{0,1\}^n$ 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 (NeuralR2RO), 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 increasing 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. NeuralR2RO 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 reduction and 5 to 10 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, x, are referred to as here-and-now decisions and are made before the uncertainty is realized. The second set of decisions, y, is referred to as the wait-and-see decisions and can be made on the fly after the uncertainty is realized. The uncertain parameters are assumed to be contained in a convex and bounded uncertainty set $\Xi \subseteq \mathbb{R}^q$. The 2RO problem aims at finding a first-stage solution x which minimizes the worst-case objective value over all scenarios $\xi \in \Xi$, where for each scenario the best possible second-stage decision $y(\xi)$ is implemented. Mathematically, a 2RO problem is given by

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

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

where $X \subseteq \mathbb{R}^n$ and $Y \subseteq \mathbb{R}^m$ are feasible sets for the first- and second-stage decisions, respectively. In this work, we consider the challenging case of integer sets X and Y. All parameters of the problem, namely $c(\cdot) \in \mathbb{R}^n$; $d(\cdot) \in \mathbb{R}^m$; $W(\cdot) \in \mathbb{R}^{r \times m}$; $T(\cdot) \in \mathbb{R}^{r \times n}$, and $h(\cdot) \in \mathbb{R}^r$ depend on the scenario. 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 $x \in X$, we have a method that calculates a scenario $\zeta \in \Omega$ for which the second-stage constraints $T(\zeta)x + W(\zeta)y \geq h(\zeta)$ over $y \in Y$ 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 $\Omega^0 \subseteq \Omega$. The latter problem can be modeled as a MILP by introducing copies of the second-stage decision variables y for each of the scenarios in Ω^0. 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 a 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 that 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.

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 y is introduced. When a large number of scenarios is required to obtain

Figure 1: Column-and-constraint generation: in each iteration, a *main problem* (box (a)) is solved to find a good first-stage solution $x^\star$ 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 $x^\star$ is not feasible anymore in MP. If no such scenario exists, then $x^\star$ 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)).

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 sum of the first- and second-stage problem for a given input of a first-stage decision, an uncertainty realization, and the problem's specification, 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

$$NN_\Theta(x, \xi; P) \approx \min_{y \in Y} \{ c_P(\xi)^\intercal x + d_P(\xi)^\intercal y : W_P(\xi) y \geq h_P(\xi) - T_P(\xi) x \}, \tag{3}$$

where Θ are the weights of the neural network. As an alternative, since $c_P(\xi)^\intercal x$ is just a simple scalar product of the input vectors, we instead could only predict the second-stage objective, i.e., $d_P(\xi)^\intercal y$, subject to the same constraints. However, as discussed later in this section and Appendix H.3, predicting the sum of first- and second-stage objectives achieves higher-quality solutions.

For ease of notation, we hereafter omit P in the formulation. For more details on the architecture of $NN_\Theta(\cdot)$, see Section 3.3.

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.

$$\min_{x \in X, y \in Y, \xi_a \in \Xi} \quad c(\xi_a)^\intercal x + d(\xi_a)^\intercal y \tag{4a}$$

$$\text{s.t.} \quad W(\xi_a) y + T(\xi_a) x \geq h(\xi_a); \tag{4b}$$

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

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 bilinear integer programming formulation, or MIP, where we refer to MIP rather than MILP for mixed-integer non-linear programs. The MP loses its linearity by introducing the ξ_a variable.

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, (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 second-stage cost for each scenario, we only require that the neural network be able to identify the maximal scenario. Prediction inaccuracy is then compensated for in (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 max and max formulations on the knapsack problem, and establishes that the max formulation indeed computes higher quality solutions across every instance.

Adversarial Problem. In the AP of Neur2RO, we replace the inner optimization problem over y by its prediction $NN(x^?;)$, where $x^?$ is given (see Figure 1 box (d)). If the first-stage objective does not depend on the uncertainty parameter, we can omit the first stage from the prediction, as neither the arg max term in the master problem nor the AP depends on it.

When we deal with constraint uncertainty, we first check if there exists a scenario , such that no feasible $y \in Y$ exists for the constraints

$$W( )y + T( )x^? \quad h( );$$

which we can do by the assumption from Section 2.1. If such a scenario exists, we add to ^{0} and continue with solving the MP again. Note that in this case $x^?$ is not feasible for MP in the next iteration. If no such scenario could be found, we calculate an optimal solution $^?$ of the AP in box (d) of Figure 1 which can be done by using the MILP representation for NN. Note that if 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

$$\max_{2} NN(x^?;) \quad \max_{2 \ ^0} NN(x^?;) + " \qquad (5)$$

for a pre-defined accuracy parameter $" > 0$, then we add $^?$ to ^{0} 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 ^{0} before we iterate to the MP.

As the adversarial problem requires finding the worst-case uncertainty over the input of a neural network, heuristic approaches may provide a significant benefit in terms of solving time at minimal degradation in solution quality. As the uncertainty set is often polyhedral, a general heuristic in the case of a continuous uncertainty set would be either sampling or using the LP relaxation of the MILP representation of NN, which are explored computationally in Appendix H.2 and H.2.2, respectively. In general, both approaches provide improved efficacy at the cost of worse solution quality. Note, when applying a heuristic to obtain a solution of the AP, we afterwards evaluate the AP objective value by a forward-pass of the trained NN to obtain the precise objective value, to ensure that the stopping criterion (5) is still valid. Since we do not necessarily find the worst-case scenario by a heuristic approach, our algorithm may terminate earlier in this case.

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 X is bounded; this indeed holds for the knapsack and capital budgeting problems.

Theorem 1. If X is finite, the ML-based CCG terminates after a finite number of iterations.

Discussion of the target. As mentioned in Section 3.1, either the sum of first- and second-stage objectives or just the second-stage objective may be predicted. When predicting only the second-stage objective, the first-stage cost must be included in the MP and AP (see Appendix H.3 for the exact formulation). Empirically, Appendix H.3 demonstrates that predicting the sum of the first- and second-stage objectives yields significantly better solutions. On the methodological side, when only the second stage is predicted each node in the branch-and-bound tree being explored by a MIP solver will contain the exact first-stage and the predicted second-stage objectives. As such, we speculate

that the LP relaxation at each node will consist of two components that are on entirely different scales. Speci cally, the rst-stage objective will be tight as it is being represented exactly while the second-stage objective requires the relaxation of the prediction model which will not be tight due to the big-M constraints. This means that the maximization problem in the AP favors the second stage. This mismatch could lead to inaccurate scenarios and undesirable downstream effects within branch-and-bound.

3.3 ARCHITECTURE

Figure 2: Neural network architecture for ML-based CCG. The current rst-stage solution, is embedded once using the network ( ). A scenario is embedded using the network ( ). To estimate the value of the second-stage optimization problem corresponding to a particular pair ( ), the two embedding vectors are concatenated into one (dashed arrows) and then passed into the nal Value Network

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 (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 ef ciently optimized over is a crucial aspect of an ef cient ML-based CCG algorithm.

To achieve ef cient optimization, we embed the rst-stage decisions, and a scenario, , into low-dimensional embeddings using networks and , respectively. These embeddings are concatenated and passed through a nal small neural network (the Value Network) 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 a MIP, we only have to represent the embedding network and the small value network, 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 ( ) can be precomputed via a forward pass for each scenario, i.e., no MILP representation is needed for . If is a small neural network, then representing a large number of copies of the network (one per scenario) remains amenable to ef cient optimization.

Adversarial Problem Optimization. For the AP, we only require representing and as the embedding of 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 coef cients, and objective coef cients of decision variables is required. To handle this, 2RO leverages set-based neural networks Zaheer et al. (2017), for and . Speci cally, embeddings are computed for each single rst-stage and scenario variable ( ) using their values, constraints, and objective coef cients, via a network with shared parameters. These embeddings are then aggregated and passed through an additional feed-forward neural network to derive the rst-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 was used for all learning models (Paszke et al., 2019). Gurobi 10.0.2 (Gurobi Optimization, LLC, 2023) was used as the MIP 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 train a single set-based architecture for each 2RO problem (see Appendix D for details of architecture). For optimization in 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 3.2 for the MP. Additionally, we terminate solving the MP or AP early if no improvement in the solution is observed in 180 seconds. For further experiments with other variants of Neur2RO, see Appendix H.

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 size $n \in \{20, 30, 40, 50, 60, 70, 80\}$.
- Capital budgeting. These problem instances are generated similar to Subramanyam et al. (2020).While uncertain parameters appear in the constraints (see (1c)), we can easily verify the assumption given in Section 2.1 as follows: for every y we check if $\max_{\xi \in \Xi} c(\xi)^\top y \leq B$, 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 always choose $z = 0$. On the other hand, if the inequality is violated, then no feasible second-stage solution exists since $y \neq 0$, 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 $x^?$, we obtain the corresponding objective value by solving (2) for a fixed x. For knapsack, this can be efficiently solved by constraint generation on y. For capital budgeting on the other hand, due to constraint uncertainty, we cannot use the constraint generation approach. Instead we sample scenarios ξ and solve (2) with fixed x and ξ. 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 $\text{obj}^?$ 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 $100 \cdot \frac{|\text{obj}^? - \text{obj}_A|}{|\text{obj}^?|}$. 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 once both x and ξ 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.

For training, we train one size-independent model for each problem for 500 epochs. All times are reported in Table 1. 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.

Problem	Data Collection	Training	Total
Knapsack	2,162	2,978	5,141
Capital budgeting	3,178	2,657	5,836

Table 1: Data collection, training, and total times. Data collection is parallelized over 16 processes. All times in seconds.

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. Tables 2-3 report the median RE and solving times. In addition, detailed distributional information in the form of box plots and other metrics are provided in Appendix G.

Correlation Type	# items	Median RE		Times	
		Neur2RO	BP	Neur2RO	BP
Uncorrelated	20	1.417	0.000	7	0
	30	1.188	0.000	9	1
	40	1.614	0.000	13	3
	50	1.814	0.000	14	12
	60	1.146	0.000	24	18
	70	1.408	0.000	27	46
	80	0.994	0.000	20	388
Weakly Correlated	20	1.705	0.000	7	29
	30	2.236	0.000	16	454
	40	1.667	0.000	45	6,179
	50	1.756	0.000	42	8,465
	60	0.772	0.000	134	9,242
	70	0.068	0.020	32	10,800
	80	0.000	0.345	45	10,800
Almost Strongly Correlated	20	1.798	0.000	7	9
	30	0.627	0.000	10	2,708
	40	0.497	0.000	17	4,744
	50	0.019	0.000	13	8,852
	60	0.047	0.000	27	10,261
	70	0.031	0.031	34	10,800
	80	0.106	0.035	26	10,800
Strongly Correlated	20	1.774	0.000	8	9
	30	0.670	0.000	11	2473
	40	0.542	0.000	20	5,665
	50	0.073	0.000	18	8,240
	60	0.000	0.046	21	10,800
	70	0.020	0.027	28	10,800
	80	0.000	0.032	31	10,800

Table 2: 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.

Knapsack. Table 2 demonstrates a clear improvement in scalability, with the solving time of Neur2RO ranging between 7 and 134 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 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

# items	Median RE				Times (seconds)			
	Neur2RO	k = 2	k = 5	k = 10	Neur2RO	k = 2	k = 5	k = 10
10	0.894	1.140	0.000	0.000	112	20	9,561	10,800
20	0.000	0.199	0.110	0.085	436	8,702	10,800	10,800
30	0.065	0.018	0.073	0.034	851	10,801	10,800	10,800
40	0.005	0.076	0.011	0.019	1,153	10,806	10,801	10,801
50	0.000	0.041	0.031	0.020	1,304	10,807	10,804	10,801

Table 3: Median RE and solving times 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.

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 G 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 5 to 10 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 0.065 compared to the best baseline's 0.018; solutions that are within 0.065% 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 3, 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-derived 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), and stochastic programming (Dumouchelle et al., 2022; Kronqvist et al., 2023). 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 finds 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 adversarial problem in the column-and-constraint generation algorithm optimizes over the inputs of a neural network, which could benefit from the many "adversarial attack" heuristics in the ML literature. We have experimented with solving only the linear relaxation of the neural network MILP representation and sampling of scenarios as an alternative to solving the adversarial problem exactly. However, we speculate that better heuristics for adversarial attacks may lead to even stronger performance. Generalizations of Neur2RO that make predictions of the infeasibility of a particular constraint could be explored. In terms of the MILP representation of our neural networks, we have used the most basic big-M formulation but stronger ones such as those offered in the package OMLT (Ceccon et al., 2022) may bring about improved solution times. Additionally, ReLU networks are not the only models that are MILP-representable; decision trees (Bertsimas & Dunn, 2017) or rule lists (Rudin & Ertekin, 2018) could be used as alternatives and may be appropriate for other problem settings.

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 Jack Dunn. Optimal classification trees. Machine Learning, 106:1039–1082, 2017.

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.

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 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. 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.

Cynthia Rudin and Şeyda Ertekin. Learning customized and optimized lists of rules with mathematical programming. *Mathematical Programming Computation*, 10:659–702, 2018.

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.

Ihsan 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.

A 2RO PROBLEMS

A.1 ROBUST TWO-STAGE KNAPSACK

We consider the two-stage knapsack problem as defined in Arslan & Detienne (2022) with a set of n items. Each item i has a weight c_i and an uncertain profit $p_i(\zeta) = p_i + \zeta_i \hat{p}_i$, where p_i is the expected profit, $\hat{p}_i$ its maximum deviation and ζ_i the uncertain profit degradation factor, where the degradation happens after the first stage. In this problem we have a budgeted uncertainty set $\mathcal{Z} = \{\zeta \in [0,1]^n : \sum_{i=1}^{n} \zeta_i \leq \Gamma\}$. The first stage decision is to choose a subset of items to produce. Then in the second stage, there are three different responses to the profit degradation: (i) accept the degraded profit, (ii) repair the item by using an additional t_i units from the budget to recover the original profit p_i, or (iii) outsource the item for a cost of f_i units, such that the item's profit results in $p_i - f_i$. This gives the following problem formulation:

$$\min_{x \in \{0,1\}^n} \max_{\zeta \in \mathcal{Z}} \min_{y \in \{0,1\}^n, r \in \{0,1\}^n} \sum_{i=1}^{n} (f_i - p_i)x_i + (p_i + \hat{p}_i\zeta_i - f_i)y_i - \hat{p}_i\zeta_i r_i$$

$$\text{s.t.} \sum_{i=1}^{n} c_i y_i + t_i r_i \leq C$$

$$r_i \leq y_i \leq x_i \qquad \forall i \in \{1,\dots,n\},$$

where x_i is the first-stage decision to produce item i. For the second-stage decisions, we have y_i and r_i: (i) $y_i = 1$ if item i is produced without repairing and $y_i = 0$ if the item is outsourced, and (ii) r_i is the decision for repairing item i.

A.2 CAPITAL BUDGETING

Consider the capital budgeting problem in Subramanyam et al. (2020), where a company aims to invest in a subset of n projects. For each project i, the uncertain cost, and profit are respectively defined as

$$c_i(\zeta) = 1 + \tfrac{1}{2}\zeta_i^\top c_i \quad \text{and} \quad r_i(\zeta) = 1 + \tfrac{1}{2}\zeta_i^\top r_i, \quad \forall i \in \{1,\dots,n\},$$

where c_i and r_i are the nominal cost and nominal profit of project i, $\zeta_i^\top$ and $\zeta_i^\top$ are the i-th row vectors of the sensitivity matrices; $\zeta \in \mathbb{R}^{N \times 4}$, with $\zeta \in [-1,1]^4$. We use the problem formulation described in (1).

B 2RO ALGORITHMS

In this section, we describe the column-and-constraint generation algorithm in more detail and the k-adaptability problem, briefly describing one of its solution methods.

B.1 COLUMN-AND-CONSTRAINT GENERATION

The CCG iterates between the main problem (MP) and the adversarial problem (AP). The MP is given as

$$\min_{x \in X} \max_{\zeta \in \mathcal{Z}^0} \min_{y \in Y} \; c(\zeta)^\top x + d(\zeta)^\top y \tag{6a}$$

$$\text{s.t.} \; T(\zeta)x + W(\zeta)y \leq h(\zeta); \tag{6b}$$

where $\mathcal{Z}^0$ is a finite subset of scenarios. Clearly, the MP provides a lower bound on the optimal value of (2). To solve the MP, for each scenario in $\mathcal{Z}^0$, a copy of the second-stage variables is generated. Using a level-set transformation, the problem can be formulated as

$$\min_{x \in X} \tag{7a}$$

$$\text{s.t.} \; c(\zeta)^\top x + d(\zeta)^\top y \leq \quad \forall \zeta \in \mathcal{Z}^0 \tag{7b}$$

$$T(\zeta)x + W(\zeta)y \leq h(\zeta) \quad \forall \zeta \in \mathcal{Z}^0 \tag{7c}$$

$$\in \mathbb{R}; y \in Y \quad \forall \zeta \in \mathcal{Z}^0; \tag{7d}$$

which is a linear integer problem that state-of-the-art solvers, such as Gurobi, can solve. In each iteration of the CCG an optimal solution (x^*, η^*) of (7) is calculated. Afterwards, the AP is solved, which is de ned as

$$\max_{\in\Xi} \min_{y\in Y} \quad c(\zeta)^\top x^* + d(\zeta)^\top y \tag{8a}$$

$$\text{s.t.} \quad W(\zeta)y \geq h(\zeta) - T(\zeta)x^*. \tag{8b}$$

Since the optimal value of the AP is the objective value of the current solution x^*, it provides an upper bound on the optimal value of (2). We de ne the optimal value to be equal to in nity if there exists a scenario $\in \Xi$ for which no feasible second-stage solution y exists. If the optimal value of the AP is larger than η^* then we add the optimal scenario to Ξ^0 and start again from solving MP. Otherwise, we stop the algorithm since the upper bound is smaller or equal to the lower bound, and hence x^* is an optimal solution. The whole procedure is presented in Algorithm 1.

Algorithm 1 Column-and-Constraint Generation

```
set ub = 1, lb = −1
Ξ^0 = {ζ_0} for any ζ_0 ∈ Ξ
while ub − lb > 0 do
    Calculate an optimal solution (x*, η*) of the main problem (7) and set lb = η*.
    Calculate an optimal solution ζ* (with optimal value opt) of the adversarial problem (8) where
    x = x*.
    Set Ξ^0 = Ξ^0 ∪ {ζ*} and ub = min{ub, opt}.
end while
return x*
```

CCG often fails to calculate an optimal solution in a reasonable time since both the MP and the AP are very hard to solve in the case of integer second-stage variables. In each iteration, the size of MP increases since we have to add new constraints and a copy of all integer second-stage decisions y. This often leads to the situation that after even a small number of iterations, the MP cannot be solved to optimality anymore by classical integer optimization solvers as Gurobi.

Furthermore, solving the AP is extremely challenging for integer second-stage variables. Indeed, the problem can be formulated as a bilevel problem where the follower problem contains integer variables. In Zhao & Zeng (2012) the authors present a column-and-constraint algorithm that solves the AP if the second stage is a mixed-integer problem. One drawback is that this method is not applicable if the second stage does not contain continuous variables, as is the case for many problems, e.g., the capital budgeting problem. Furthermore, the method involves solving a very large mixed-integer bilinear problem, which is computationally enormously challenging. The whole procedure must be executed in each iteration of the main CCG algorithm.

B.2 k-ADAPTABILITY

The k-adaptability approach was introduced in Bertsimas & Caramanis (2010) and later studied for objective uncertainty and constraint uncertainty in Hanasusanto et al. (2015); Subramanyam et al. (2020); Ghahtarani et al. (2023); Julien et al. (2022); Kurtz (2023). The main idea of the approach is to calculate a set of k second-stage solutions already in the first stage. Instead of choosing the best feasible second-stage solution for each scenario, we choose the best of these k calculated second-stage solutions. Since we restrict the number of second-stage reactions, this approach leads to feasible solutions of (2), which are not necessarily optimal. While for larger k the approximation guarantee gets provably better, the problem gets harder to solve at the same time. Furthermore, it was shown in Subramanyam et al. (2020) that it may happen k that has to be chosen exponentially large to guarantee optimality for (2). The k-adaptability problem can be formulated as

$$\min_{x\in X,\, y^1,\ldots,y^k\in Y} \quad \max_{\in\Xi} \quad \min_{y\in\{y^1,\ldots,y^k\}} \quad c(\zeta)^\top x + d(\zeta)^\top y \tag{9a}$$

$$\text{s.t.} \quad W(\zeta)y + T(\zeta)x \geq h(\zeta). \tag{9b}$$

The k-adaptability problem is very challenging to solve, especially in the constraint uncertainty case. The best-known method for this case was introduced in Subramanyam et al. (2020). The authors

perform a branch-and-bound algorithm over partitions of the uncertainty set. They consider partitions of finite scenario sets, which are iteratively generated, and assign each of the second-stage solutions to one of the partitions. This approach was later improved by applying machine learning methods to improve the branching decisions Julien et al. (2022). As an alternative approach in Postek & Hertog (2016), an iterative uncertainty set splitting method is presented, which converges to the exact optimal value of the two-stage robust problem.

In case of objective uncertainty, the adaptablity problem is easier (but still hard) to solve (Arslan et al. (2022); Ghahtarani et al. (2023)) and can be approximated if K is not too small; see Kurtz (2023).

C Detailed Formulation

This section presents the detailed big-M formulation for (4). We assume that at this iteration in the MP, we have scenarios $\xi_1, \dots, \xi_k$ and that M and L are upper and lower bounds on the prediction of the network. The complete formulation is then given by

$$\min_{x \in X,\, y \in Y,\, \alpha \in \mathbb{R},\, p, u, z \in \{0,1\}^k} \quad c(\alpha)^\top x + d(\alpha)^\top y \tag{10a}$$

$$\text{s.t.} \quad W(\alpha) y + T(\alpha) x \geq h(\alpha); \tag{10b}$$

$$p_i = NN(x, \xi_i) \qquad \forall i \in \{1, \dots, k\} \tag{10c}$$

$$u \geq p_i \qquad \forall i \in \{1, \dots, k\} \tag{10d}$$

$$u \leq p_i + (M - L)(1 - z_i) \qquad \forall i \in \{1, \dots, k\} \tag{10e}$$

$$\sum_{i=1}^{k} z_i = 1 \qquad \forall i \in \{1, \dots, k\} \tag{10f}$$

$$\alpha = \sum_{i=1}^{k} z_i \xi_i \tag{10g}$$

To model the arg max, we introduce k binary variables z and $k+1$ continuous variables p and u, which are used to model big-M that ensures z is 1 at the index of the maximizer and 0 everywhere else. α is then given by a linear combination of the scenarios multiplied with z.

D Extended NN Architecture

We show the extended neural network architecture used in the experiments in Figure 3.

E 2RO with fixed first-stage decision

When we compare the calculated solutions of our 2RO with the baseline in our experiments, we need to calculate the objective value of a solution $x^? \in X$ exactly or approximately. The former involves solving the AP (8) for a given solution. Solving this problem is intractable when we have uncertain parameters in the constraints. We first expand on how the adversarial would be solved in a tractable way if the uncertain parameters only appear in the objective function. Subsequently, we describe an approach to approximately solve the AP, which is based on sampling scenarios from

E.1 Objective uncertainty

For the special case of objective uncertainty, the AP can be solved much more efficiently. In this case, the adversarial problem is given as

$$\max_{\alpha \in \mathcal{A}} \min_{y \in Y} \quad c(\alpha)^\top x^? + d(\alpha)^\top y \tag{11a}$$

$$\text{s.t.} \quad Wy \geq h - Tx^?; \tag{11b}$$

Figure 3: The extended neural network architecture for ML-based CCG. Compared to the NN architecture shown in the main text (Figure 2), this model uses the set-based architecture to generalize across instance sizes. Let $x^? \in R^n$ be the candidate first-stage decision (found in the MP) and $^k \in R^q$ the scenario found in the k-th iteration in the AP. Then, $\hat{}_x$ and $\hat{}$ are the embedding networks for $x_i; i \in [n]$ and $_j; j \in [q]$, respectively. The features are comprised of the single variable and single-variable specific problem specification $P^i_x; i \in [n]$ and $P^j; j \in [q]$ for first-stage decisions and scenarios, respectively. The outputs of the networks are aggregated for x and ^{k} separately. These embeddings are the input of the original NN given in the main part.

which can be reformulated as

$$\max_{\in} \quad \tag{12a}$$

$$\text{s.t.} \quad c(\ )^| x^? + d(\ )^| y \quad 8y \in Y; \tag{12b}$$

where $Y = \{y \in Y : Wy \quad h \quad Tx^?\}$. While the set Y can contain an exponential number of solutions, the latter problem can be solved by iteratively generating the constraints for $y \in Y$.

E.2 Constraint uncertainty

We collect all scenarios $\in$ which were generated during training and during the solution procedures of the baseline algorithm and our algorithm (including the scenarios calculated by the AP) in the set $^{\text{samples}}$. Then for the two returned solutions x^s and x^{baseline} we compare

$$\max_{\in} \min_{y \in Y} \quad c(\ )^| x^? + d(\ )^| y \tag{13a}$$

$$\text{s.t.} \quad W(\ )y \quad h(\ ) \quad T(\ )x^?; \tag{13b}$$

where we replace x by the corresponding solution x^s or x^{baseline}. The latter problem can be solved by calculating the optimal value of the second-stage problem for each scenario independently and choosing the worst-case overall optimal values.

F Convergence

In the following, we present the proof of Theorem 1.

Proof. The main idea is to show that condition (5) cannot hold in infinitely many iterations. Since we stop the algorithm if (5) is not true anymore, then finite termination of the algorithm follows.

Assume the algorithm does not terminate in a finite number of iterations. Let l_t and r_t be the values of the left-hand side and right-hand side of inequality (5) in iteration t of the algorithm, i.e.,

$$l_t := \max_{\in} NN \quad (x^t; \ )$$

and

$$r_t := \max_{\in\ ^t} NN \quad (x^t; \ ):$$

where x^t is the optimal solution of MP in the t-th iteration and Ω^t the finite set of scenarios used in the MP in iteration t. Let $x \in X$ be a feasible first-stage solution and let $l_t(x)$ and $r_t(x)$ be the sub-sequences which contain the values of s_t and r_t only for the iterations where x is an optimal solution of the MP. Then either this sequence is finite or, if it is infinite, the sequence $(r_t(x))_t$ is monotonous and bounded where monotony follows since $\Omega^t \subseteq \Omega^{t+1}$ and since the same x is used. The sequence is bounded since Ω a bounded set and NN a piecewise-linear function (as is known for feedforward ReLU networks (Montufar et al., 2014)) and the maximum of a piecewise linear function over a bounded set is bounded. Hence $(r_t(x))_t$ converges to a finite value $r^?(x)$. Furthermore, it holds $s_t(x) \geq r_{t+1}(x)$ since the optimal scenario of the left-hand-side is added to Ω^t which is a subset of the set later used to evaluate $r_{t+1}(x)$. It follows that

$$r_t(x) - l_t(x) > \varepsilon \quad \Rightarrow \quad r_{t+1}(x) - l_t(x) > \varepsilon$$

for all t which contradicts the convergence of $(r_t(x))_t$. Hence the sequence $(r_t(x))_t$ must be finite. Since only finitely many first-stage solutions x exist, and the latter result holds for all of them, the number of iterations of the algorithm must be finite. □

G Distributional Results for Relative Performance

In this section, we provide distributional information for the RE for knapsack in Tables 4-5 and Figures 4-G.

Correlation Type	# items	Mean RE		Median RE		RE 1st Quartile		RE 3rd Quartile	
		Neur2RO	BP	Neur2RO	BP	Neur2RO	BP	Neur2RO	BP
Uncorrelated	20	2.005	0.000	1.417	0.000	0.541	0.000	2.379	0.000
	30	1.189	0.000	1.188	0.000	0.712	0.000	1.399	0.000
	40	2.895	0.000	1.614	0.000	1.221	0.000	4.042	0.000
	50	3.055	0.000	1.814	0.000	0.946	0.000	3.801	0.000
	60	2.106	0.000	1.146	0.000	0.577	0.000	2.872	0.000
	70	2.228	0.000	1.408	0.000	0.761	0.000	2.514	0.000
	80	1.595	0.000	0.994	0.000	0.758	0.000	2.063	0.000
Weakly Correlated	20	2.882	0.000	1.705	0.000	1.260	0.000	4.983	0.000
	30	2.571	0.000	2.236	0.000	0.616	0.000	3.921	0.000
	40	2.508	0.000	1.667	0.000	1.180	0.000	2.234	0.000
	50	2.174	0.137	1.756	0.000	0.793	0.000	3.010	0.000
	60	2.118	0.365	0.772	0.000	0.000	0.000	1.576	0.458
	70	0.813	0.338	0.068	0.020	0.000	0.000	0.715	0.175
	80	0.390	0.690	0.000	0.345	0.000	0.000	0.239	0.738
Almost Strongly Correlated	20	2.659	0.000	1.798	0.000	0.072	0.000	3.006	0.000
	30	1.055	0.113	0.627	0.000	0.054	0.000	1.538	0.000
	40	0.796	0.381	0.497	0.000	0.019	0.000	1.563	0.000
	50	0.457	0.896	0.019	0.000	0.000	0.000	0.673	1.396
	60	0.214	0.454	0.047	0.000	0.000	0.000	0.279	0.767
	70	0.521	0.600	0.031	0.031	0.000	0.000	0.917	1.159
	80	0.426	0.658	0.106	0.035	0.000	0.000	0.543	0.770
Strongly Correlated	20	2.548	0.000	1.774	0.000	0.905	0.000	3.190	0.000
	30	1.137	0.121	0.670	0.000	0.054	0.000	1.847	0.000
	40	0.672	0.191	0.542	0.000	0.002	0.000	0.888	0.000
	50	0.413	0.648	0.073	0.000	0.000	0.000	0.821	0.963
	60	0.313	0.379	0.000	0.046	0.000	0.000	0.241	0.540
	70	0.485	0.721	0.020	0.027	0.000	0.000	0.668	0.793
	80	0.363	0.638	0.000	0.032	0.000	0.000	0.464	0.906

Table 4: Table of distributional information for knapsack. For each row, all RE statistics are computed over 18 instances.

# items	Mean RE				Median RE				RE 1st Quartile				RE 3rd Quartile			
	Neur2RO	k = 2	k = 5	k = 10	Neur2RO	k = 2	k = 5	k = 10	Neur2RO	k = 2	k = 5	k = 10	Neur2RO	k = 2	k = 5	k = 10
10	2.396	2.925	1.021	1.130	0.894	1.140	0.000	0.000	0.000	0.000	0.000	0.000	3.287	4.829	0.547	1.512
20	0.326	0.300	0.234	0.270	0.000	0.199	0.110	0.085	0.000	0.026	0.016	0.009	0.313	0.436	0.325	0.362
30	0.459	0.148	0.131	0.083	0.065	0.018	0.073	0.034	0.001	0.000	0.001	0.000	0.291	0.182	0.212	0.080
40	0.103	0.114	0.098	0.073	0.005	0.076	0.011	0.019	0.000	0.001	0.000	0.002	0.095	0.180	0.137	0.137
50	0.032	0.106	0.089	0.056	0.000	0.041	0.031	0.020	0.000	0.001	0.000	0.001	0.019	0.191	0.138	0.081

Table 5: Table of distributional information for capital budgeting. For each row, all RE statistics are computed over 50 instances.

Figure 4: Box plot of RE for baseline and Neur2RO on UN knapsack instances.

Figure 5: Box plot of RE for baseline and Neur2RO on WC knapsack instances.

Figure 6: Box plot of RE for baseline and Neur2RO on ASC knapsack instances.

H Ablation

This section presents an ablation across three aspects of Neur2RO, namely, the formulation of the MP, the method to obtain worst-case scenarios, and the prediction target for problems with objective uncertainty.

Figure 7: Box plot of RE for baseline and Neur2RO on SC knapsack instances.

Figure 8: Box plot of RE for baselines and Neur2RO on capital budgeting instances.

H.1 Main Problem Formulation

As an alternative to the formulation using arg max over a set of scenarios. One more straightforward formulation is to consider instead the max over all of the scenarios, which is given by

$$\min_{x \in X,\ \theta \in \mathbb{R}} \quad \theta \tag{14a}$$

$$\text{s.t.} \quad NN(x, \xi_k) \geq \theta \qquad \forall k \in \{1, \ldots, K\}. \tag{14b}$$

Table 6 reports the MRE of the arg max and max formulations and the solving time for the knapsack instances. Table 6 demonstrates an improvement in solution quality, with max obtaining a lower median RE in every case and a lower computing time in most cases.

H.2 Worst-case Scenario Acquisition

This section compares the adversarial approach for determining scenarios to a sampling and an LP relaxation-based approach.

H.2.1 Sampling-Based Scenario Acquisition

For sampling, as a baseline, we sample 100,000 scenarios, and then to approximate the AP, we take the maximizer over a forward pass. Table 7 demonstrates a clear trade-off between solution quality and efficiency. Generally, sampling improves average solving time across all instances but leads to worse solution quality as the instance size increases.

H.2.2 LP Relaxation-Based Scenario Acquisition

For the LP relaxation, we compare the performance of using the standard MILP-based scenario acquisition (standard), i.e., solving the AP to optimality, to the relaxation (LP relaxation). For both problems, we report the RE to the baselines. Tables 8 and 9 present the results for knapsack and

Correlation Type	# items	Median RE arg max	Median RE max	Times arg max	Times max
Uncorrelated	20	0.000	1.167	7	16
	30	0.000	0.945	9	21
	40	0.000	1.931	13	37
	50	0.000	1.488	14	49
	60	0.000	0.452	24	56
	70	0.000	0.801	27	53
	80	0.000	2.227	20	58
Weakly Correlated	20	0.000	3.515	7	22
	30	0.000	2.405	16	37
	40	0.000	0.455	45	59
	50	0.000	0.254	42	73
	60	0.000	1.528	134	100
	70	0.000	1.769	32	57
	80	0.000	3.492	45	85
Almost Strongly Correlated	20	0.000	2.042	7	18
	30	0.000	1.433	10	23
	40	0.000	1.739	17	51
	50	0.000	3.161	13	34
	60	0.000	2.449	27	48
	70	0.000	2.497	34	52
	80	0.000	1.824	26	53
Strongly Correlated	20	0.000	1.154	8	18
	30	0.000	0.967	10	23
	40	0.000	1.911	19	45
	50	0.000	3.613	15	34
	60	0.000	2.005	24	42
	70	0.000	2.657	31	49
	80	0.000	2.051	28	48

Table 6: arg max and max formulations on knapsack instances. For each row, the median RE and solving time are computed over 18 instances. All times in seconds.

capital budgeting, respectively. In general, we can observe that the LP relaxation does indeed lead to signi cantly faster solving time, with an overall decreased solution quality. That being said, for capital budgeting in particular, Neur2RO with the LP relaxation still achieves a lower median RE than the baselines on larger instances, while being roughly ve times faster than results without the relaxation.

H.3 Prediction Target

This section compares the prediction target. For capital budgeting, the coef cients of the rst-stage decisions in the objective contain uncertainty. As such, this presents a choice of either predicting the sum of the rst- and second-stage objectives, i.e., $c(\cdot)^\top x + \min_{y \in Y} d(\cdot)^\top y : W(\cdot)y \geq h(\cdot) - T(\cdot)x$, or only the second-stage objective, i.e., $\min_{y \in Y} d(\cdot)^\top y : W(\cdot)y \geq h(\cdot) - T(\cdot)x$. Speci cally, we compare the downstream optimization performance with respect to the resulting formulations. The formulation for predicting the sum of the rst- and second-stage objectives is presented in 3.2. For predicting the second-stage objective only, the MP is given by

$$\min_{x \in X,\, y \in Y,\, \alpha \in \mathbb{R}} \quad c(\alpha)^\top x + d(\alpha)^\top y \tag{15a}$$

$$\text{s.t.} \quad W(\alpha)y + T(\alpha)x \geq h(\alpha); \tag{15b}$$

$$\alpha \in \arg\max_{\beta \geq 0} c(\beta)^\top x + NN_\theta(x, \beta); \tag{15c}$$

and the AP is given by

$$\max_{\beta} c(\beta)^\top x^? + NN_\theta(x^?, \beta). \tag{16}$$

The main difference with this formulation is that the objective coef cients $c(\cdot)$ can be utilized directly rather than requiring the ML model to predict them. Table 10 compares the two approaches on the capital budgeting instances wherein the RE is computed with respect to the baselines. From the table, both solution quality and time are greatly improved when predicting both rst- and second-stage objectives.

Correlation Type	# items	Median RE		Times	
		adversarial	sampling	adversarial	sampling
Uncorrelated	20	**0.000**	**0.000**	7	**3**
	30	**0.000**	**0.000**	9	**4**
	40	0.560	**0.000**	13	**6**
	50	0.723	**0.000**	14	**7**
	60	0.066	**0.000**	24	**8**
	70	0.295	**0.000**	27	**9**
	80	0.182	**0.000**	20	**11**
Weakly Correlated	20	**0.000**	0.074	7	**3**
	30	**0.000**	0.444	16	**6**
	40	**0.000**	0.093	45	**7**
	50	0.441	**0.000**	42	**8**
	60	**0.000**	0.250	134	**12**
	70	**0.000**	0.185	32	**12**
	80	**0.000**	0.536	45	**13**
Almost Strongly Correlated	20	**0.000**	**0.000**	7	**3**
	30	**0.000**	0.402	10	**5**
	40	**0.000**	0.663	17	**7**
	50	**0.000**	0.587	13	**7**
	60	**0.000**	0.690	27	**9**
	70	**0.000**	0.111	34	**10**
	80	**0.000**	0.827	26	**11**
Strongly Correlated	20	**0.000**	**0.000**	8	**3**
	30	**0.000**	0.402	10	**5**
	40	**0.000**	0.238	19	**7**
	50	**0.000**	0.574	15	**7**
	60	**0.000**	0.558	24	**9**
	70	**0.000**	0.174	31	**10**
	80	**0.000**	0.548	28	**11**

Table 7: Adversarial and sampling-based approaches for worst-case scenario acquisition on knapsack instances. For each row, the median RE and solving time are computed over 18 instances. All times in seconds.

Correlation Type	# items	Median RE		Times	
		standard	LP relaxation	standard	LP relaxation
Uncorrelated	20	**1.417**	1.673	7	**1**
	30	1.188	**1.167**	9	**1**
	40	1.614	**1.387**	13	**2**
	50	1.814	**1.660**	14	**2**
	60	**1.146**	**1.146**	24	**1**
	70	1.408	**1.166**	27	**2**
	80	0.994	**0.970**	20	**2**
Weakly Correlated	20	**1.705**	1.454	7	**1**
	30	**2.236**	2.034	16	**1**
	40	**1.667**	2.733	45	**2**
	50	1.756	**1.126**	42	**2**
	60	0.772	**0.729**	134	**3**
	70	**0.068**	0.243	32	**3**
	80	**0.000**	0.316	45	**9**
Almost Strongly Correlated	20	1.798	**1.211**	7	**1**
	30	**0.627**	0.665	10	**1**
	40	**0.497**	0.927	17	**2**
	50	**0.019**	1.884	13	**2**
	60	**0.047**	1.079	27	**2**
	70	0.031	**0.025**	34	**4**
	80	**0.106**	1.775	26	**4**
Strongly Correlated	20	1.774	**1.368**	8	**1**
	30	**0.670**	0.796	11	**2**
	40	**0.542**	1.375	20	**3**
	50	**0.073**	2.333	18	**2**
	60	**0.000**	0.510	21	**4**
	70	**0.020**	0.623	28	**3**
	80	**0.000**	1.097	31	**3**

Table 8: Median RE and solving times for knapsack instances with LP relaxation. 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.

I Machine Learning Model Details

I.1 Features

Here we provide the features for each of the problems. In both cases, set-based architectures Zaheer et al. (2017) with parameter sharing are utilized, so we report the features for a single dimension of the first-stage decision and scenario accordingly. Table 11 reports all of the features for each instance.

I.2 Model Hyperparameters

This section reports the hyperparameters for the neural networks for each problem. For both problems, we have the same architecture with slightly different hyperparameters. As the objective of

# items	Median RE		Times	
	standard	LP relaxation	standard	LP relaxation
10	**0.894**	2.663	112	**4**
20	**0.000**	0.060	436	**142**
30	**0.065**	0.071	851	**141**
40	**0.005**	0.007	1,153	**226**
50	**0.000**	0.001	1,304	**231**

Table 9: Median RE and solving times for capital budgeting instances with LP relaxation. 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.

# items	Median RE		Times	
	sum	second only	sum	second-only
10	**0.894**	2.424	**112**	233
20	**0.000**	0.192	**436**	1,823
30	**0.065**	0.151	**851**	3,823
40	**0.005**	0.010	**1,153**	4,062
50	**0.000**	0.005	**1,304**	7,424

Table 10: Sum and second-stage only predictions for capital budgeting instances. For each row, the median RE and solving time are computed over 50 instances. Note that in these results the RE is calculated with respect to the k-adaptability and each respective ML-approach. All times in seconds.

`Neur2RO` is to enable efficient optimization, we train small networks that can achieve a low mean absolute error value to ensure that the main and adversarial problems are tractable. For this reason, no systematic hyperparameter tuning was done. Hyperparameter optimization would likely only further improve the already strong numerical results. For both problems, we train a model for 500 epochs and compute the mean absolute error on a validation set every 10 epochs. We then use the model with the lowest reported mean absolute validation error during training for evaluation.

Table 12 reports the hyperparameters for each model. As our model generalizes across instances, which requires invariance to the order and number of decision variables, both the first-stage and scenario embedding networks are set-based architectures (Zaheer et al., 2017). We refer to Figure 3 for a refresher on the overall architecture which has the following hyperparameters. The hyperparameters "$\hat{\Phi}_{\mathbf{x}}$ dimensions" and "$\Phi_{\mathbf{x}}$ dimensions" correspond to the hidden and embedding dimensions of the first-stage embedding network. Specifically, "$\hat{\Phi}_{\mathbf{x}}$ dimensions" corresponds to the network with shared parameters that embed the representation for each first-stage decision. The last dimension of "$\hat{\Phi}_{\mathbf{x}}$ dimensions" is that of the aggregated vector. The hyperparameter "$\Phi_{\mathbf{x}}$ dimensions" corresponds to the network that takes the aggregated first-stage embedding vector as input. The last dimension of "$\Phi_{\mathbf{x}}$ dimensions" specifies the embedding dimension of the first-stage embedding network. "$\hat{\Phi}$ dimensions" and "Φ dimensions" are analogous for the scenario embedding network. "Φ dimensions" correspond to the hidden dimensions of the value network. Finally, "aggregation type" specifies the type of aggregation that combines the first-stage/scenario embeddings.

I.3 Training Curves

Figures 9-10 plot the mean absolute error at every 10 epochs during training for the training and validation data. Generally, the training and validation mean absolute error is very close, and in both problems, a relatively low mean absolute error is achieved.