Solving Bilevel Power System Problems Using Deep Convolutional Neural Networks

D. Vlah, K. Šepetanc, Student Member, IEEE and H. Pandžić, Senior Member, IEEE

Abstract—Current state-of-the-art solution techniques for solving bilevel optimization problems either assume strong problem regularity criteria or are computationally intractable. In this paper we address power system problems of bilevel structure, commonly arising after the deregulation of the power industry. Such problems are predominantly solved by converting the lower-level problem into a set of equivalent constraints using the Karush-Kuhn-Tucker optimality conditions at an expense of binary variables. Furthermore, in case the lower-level problem is nonconvex, the strong duality does not hold rendering the single-level reduction techniques inapplicable. To overcome this, we propose an effective numerical scheme based on bypassing the lower level completely using an approximation function that replicates the relevant lower level effect on the upper level. The approximation function is constructed by training a deep convolutional neural network. The numerical procedure is run iteratively to enhance the accuracy.

As a case study, the proposed method is applied to a price-maker energy storage optimal bidding problem that considers an AC power flow-based market clearing in the lower level. The results indicate that greater actual profits are achieved as compared to the less accurate DC market representation.

Index Terms—Bilevel optimization, deep convolutional neural network, optimal power flow.

I. INTRODUCTION

A. Background and Paper Scope

Deregulation and liberalization of the power sector worldwide dislodged large monopolistic power utilities, allowing for private companies to become important players in the sector. However, each of the newly created entities have their own goal, e.g. generating companies want to maximize their profit, system operators maximize the security of supply, while market operators maximize social welfare. Because of the increased number of players with conflicting objectives, they need to consider each other’s goals and objective functions when optimizing their own utility. To accommodate the interaction between own and other player’s actions, the researchers commonly resort to bilevel models, where own optimization problem (the upper-level (UL) problem) is constrained by another optimization problem (the lower-level (LL) problem). This setting assumes that the lower-level behavior is known, which is the case when considering the market operator conducting its market-clearing procedure or any other regulated entity that behaves according to some widely-known rules.

As of October 5, 2021, the IEEE Xplore database [1] indicates that three flagship IEEE Power and Energy Society journals published 182 journals with word bilevel in the title (115 such papers in IEEE Transactions on Power Systems, 45 in IEEE Transactions on Smart Grid and 22 in IEEE Transactions on Sustainable Energy). These papers cover a wide range of bilevel problems. Some of the most common topics include protection of a power system against a terrorist attack, e.g. [2], pricing schemes, e.g. [3], maintenance scheduling, e.g. [4], expansion planning, e.g. [5], or optimal bidding in one or more energy [6] or financial markets [7].

Although bilevel models have been used extensively in the literature, they often suffer from two drawbacks. The first one is related to linearization, as commonly one or more variables from the lower-level problem appear in the upper-level objective function where it usually multiplies an upper-level variable. Although in many cases this can be linearized using the strong duality theorem and some of the Karush-Kuhn-Tucker optimality conditions [8], see e.g. [9], in some cases this is not possible. In such cases the authors commonly resort to the binary expansion method (see appendix B in [10]). However, besides being an approximation, this method can result in intolerable computational times, bringing us to the second drawback, i.e. computational (in)tractability. Issues with tractability often arise when the lower-level problem is stochastic or has many inequality constraints, which results in a large number of binary variables. Some authors in such cases resort to an iterative procedure that considers the complicating dual variables in the problematic bilinear terms as parameters, and updating their values in the following iteration [11].

The aim of this paper is to present a numerical scheme based on deep convolutional neural networks paired with state of the art training procedure, for solving complex bilevel problems arising in the power systems community. As a representative of such problems, we solve a bilevel problem of optimal participation of energy storage in the day-ahead energy market. To make the problem more difficult, we assume an AC-optimal power flow (OPF)-based market clearing algorithm in the lower-level. AC OPF is a challenging type of problem to which there have been numerous simplification attempts, e.g. by convexification [12]. However, to this date there is still no known exact finite convex AC OPF formulation that classical approaches could solve to optimality, which raises the importance of development of numerical optimization schemes for bilevel optimization with nonconvex lower-level problem. Optimal bidding problems consist of two interlinked optimizations. The first problem, also called the leader or the upper-level problem, represents the market participant that maximizes its profit due to arbitrage, while the second problem, also called the follower or the lower-level problem, is the market clearing that maximizes the social welfare and determines the electricity prices that depend on the bids from the upper-level problem. The described bilevel optimization can not be directly solved using commercial of-the-shelf
solvers, thus they are usually converted into a single-level equivalent optimization problem. However, in the considered case such conversion is especially difficult since the exact AC OPF models, which are in the lower level, are nonconvex and thus render many existing techniques inapplicable.

**B. Literature Review**

As depicted in Fig. 1, the existing literature on bilevel solution techniques branches out in two main directions: classical and evolutionary approaches. Due to computational difficulty of bilevel problems, the classical approaches can only tackle well-behaved problems with strong assumptions, such as linearity or convex quadraticity and continuity, as strong duality generally does not hold for other types of problems. By far the most common classical approach is a single-level reduction based on Karush–Kuhn–Tucker (KKT) conditions and duality theory. It has been widely used to solve bilevel problems with linear constraints and linear [13] or convex quadratic [14] objective functions in either or both levels. The resulting formulations contain complementarity constraints which are combinatorial in nature and thus can be modeled using binary variables making the final problems mixed-integer linear (MILP) or mixed-integer quadratic (MIQP). The existing state-of-the-art solvers generally handle well these types of optimizations using the branch-and-bound method for binary search tree and simplex for search-tree node subproblems, despite the exponential complexity in the worst case. The single-level reduction technique has also been successfully applied to a case where the lower level is a convex quadratically-constrained quadratic problem (QCQP) as in [15], albeit, only if the interaction variables between the upper and the lower levels are discrete so that the nonlinearities can be avoided. Other classical approaches are the descent method, the penalty function method and the trust-region method. The descent method determines the most favorable variable change for the objective function, as demonstrated in [16], so that the model stays feasible. However, since the model is feasible only when the lower level is optimal, finding the descent direction is very difficult. The penalty function method replaces the lower-level [17] or both-level [18] constraints with penalty terms for constraint violations in the objective function. The trust region algorithms iteratively approximate the lower level around the operating point with linear problem (LP) or quadratic problem (QP) [19]. Both the penalty and the trust-region methods as the next step apply a KKT-based single-level reduction to the lower level and thus inherit the same applicability limits.

As opposed to the classical ones, the evolutionary approaches are inspired by the biological evolution principle where candidate solutions are evaluated using a fitness function, e.g., objective function, to form the next generation of candidate solutions by reproducing, mutating, recombining and selecting processes. Evolutionary approaches are very effective at finding good approximate solutions of numerically very difficult problems with less regularity assumptions than the classical approaches. For bilevel problems, the evolution is commonly applied in a nested form where the lower level needs to be solved separately for every upper-level solution candidate, as explained and analyzed in [20]. The upper-level solution candidates are obtained using an evolution, e.g., particle swarm optimization [21] or differential evolution [22], while the lower level can be solved using classical approaches such as interior point method [23] or as well using an evolution, as in [22]. Despite applicability to nonconvex problems, where classical approaches do not hold, the nested evolutionary method does not scale well with the number of upper-level variables as they exponentially increase the number of lower-level optimizations that need to be performed. A single-level reduction technique can also be utilized in the context of evolutionary approaches where numerical evolution concept is applied to the reduced formulation. Due to KKT conditions, this technique inherits regularity assumptions of the classical approaches for the lower-level problem, but allows a more irregular upper level. Paper [24] is one of the first works where this technique was employed. Evolutionary approaches can also be applied in tandem with the meta-modeling method. Meta-model, or surrogate model, is an easy to evaluate, typically iteratively enhanceable, approximation of the original model. For bilevel modeling, the lower level can be meta-modeled using the reaction set mapping, optimal lower-level value function, by bypassing the lower-level problem completely and by using an auxiliary bilevel meta-model. The reaction set method maps the lower-level variable values as a response to the upper-level variables. This concept was used in recent work by Sinha et al. [25]. On the other hand, the optimal lower-level function method replaces the lower objective statement of minimization or maximization with a constraint requiring that the objective is at least as good as the optimal lower-level function [26]. Both the reaction set map and the optimal lower-level function are generally difficult to obtain even in an approximated form. Bypassing the lower-level completely is based on the principle that the lower-level variables are basically functions of the upper-level variables, which allows for the upper level reformulation not to include the lower level. Similar to the trust-region method, bilevel problems can be replaced with auxiliary meta-models. As of

---

**Classical approaches**
- Single-level reduction
- Descent method
- Penalty function method
- Trust region

**Evolutionary approaches**
- Nested methods
- Single-level reduction
- Metamodeling-based methods:
  - a) reaction set mapping
  - b) optimal lower-level value function
  - c) bypassing lower-level problem
  - d) auxiliary bilevel meta-model

Fig. 1. Classification of the bilevel problem solution techniques.
current, we are not aware of any works based on bypassing the lower-level problem or the auxiliary meta-models methods. A broader bilevel solution techniques research field review, for both the classical and the evolutionary approaches, can be found in [27].

The approach presented in this paper can be classified as an evolutionary meta-modeling method that bypasses the lower-level problem completely, as given by [27]. This bypassing of the lower-level problem is achieved by approximating the solution of the lower-level, which depends only on the upper-level variables, using a carefully designed neural network (NN), see [28] and [29]. As a NN is simply a function composed of elementary functions, it can be substituted directly into the upper-level objective function. This way, the original bilevel optimization problem is reduced into a single-level optimization problem, which approximates the solution of the original problem. The main difficulty of our framework is the design and training of a NN that efficiently and accurately approximates the lower-level problem.

It is well known that for any function \( f : U \subseteq \mathbb{R}^n \rightarrow \mathbb{R}^m \) there exists a NN that uniformly approximates the given function, see [30] and [31]. What is typically not known is how exactly to construct a specific NN, approximating the function \( f \) to the desired accuracy and using the smallest possible number of neurons.

The first problem we encountered is the limited size of dataset used to train such NN. More precisely, each element in the dataset must be constructed by solving a single instance of the lower-level optimization problem, for some chosen values of the upper-level variables. Solving too many instances of the lower-level problem would take too long. On the other hand, the size of the dataset limits the maximum network size (the number of neurons), by limiting the number of parameters that define that particular network. NNs trained on a dataset that is small compared to the number of network parameters tend to overfit the training data and are poor in generalization on unseen data. In our case that would lead to lower accuracy of approximation of the lower-level problem solutions. Basically, the size of the dataset limits the accuracy of the NN approximation.

The second issue is in determining an optimal topology of a NN for a given network size, in order to achieve the greatest possible approximation accuracy. The optimal topology is dependent on an unknown function, which we are trying to approximate. Our first approach, using fully connected neural network with only few hidden layers, led to poor approximation accuracy. By carefully analyzing the properties of the lower-level optimization problem, the choice of a network topology was settled on a convolutional neural network (CNN) [32]. As the CNN architecture shares the same values of parameters between different parts of the network, the cumulative number of parameters is much smaller for the network of the same size, so the CNN architecture can be trained to approximate the original optimization problem to a higher accuracy. The first big success of the CNN architecture was in the area of computer vision, in the image classification problems [33].

The third obstacle we encountered was the generation of a dataset for the CNN training. Our first idea was to generate the dataset by uniform random sampling of the independent upper-level variables, only in intervals of their permissible values. Then for each sample, we solved the associated lower-level optimization problem. This strategy proved to be inefficient as the near-optimal values of the upper-level variables, which solve our bilevel optimization problem, are poorly represented by sampling these variables independently from the uniform distribution. It resulted in much higher approximation error of the CNN on the optimal solution than on the generated dataset. The solution proved to be in iterative refining of the generated dataset. In the first iteration we generate a uniform dataset on the whole permissible domain and find the solution of the approximation for the bilevel optimization problem. In each additional iteration we restrict the domain to an even smaller neighborhood around the approximated solution from the previous iteration. Then we generate a new uniform dataset on this smaller domain, train a new instance of the CNN, and using this new trained network, we again find a solution of the approximating problem. In each iteration we verify the quality of the current solution by computing the upper-level objective function exactly on optimal variables approximate problem values. We stop iterating when the actual value of the upper-level objective function stops improving.

C. Contribution

In this work we develop a general numerical solution technique for bilevel problems and apply it to the energy storage (ES) bidding problem on an AC-OPF-constrained energy market. The technique is applicable to any other upper-level subject, but we chose the ES due to modeling simplicity and clarity of presentation. The numerical and mathematical difficulty of solving the considered bilevel optimization arises from insufficient problem regularity due to nonconvexity of the exact AC OPF formulations. Current modeling practice is to avoid the difficulties by using a simpler linear DC OPF network representation as in [36]. To the authors knowledge, there are very few attempts to solve bilevel problems with an AC OPF in the lower level. We have not found any with the exact AC OPF, thus we single out two papers with AC OPF relaxations, [34] and [35]. Scalability and tractability issues are not discussed in these papers which is also one of the important points of this work.

The main contribution of this paper consists of the following:

1) We introduce a novel numerical scheme for solving bilevel optimization problems based on deep convolutional neural networks. It is an evolutionary meta-modeling method that completely bypasses the lower-level problem. Our method successfully works with previously intractable, i.e., nonconvex, classes of the lower-level problems. As opposed to the existing techniques, solution times are basically independent on the upper-level problem size and scale well with the lower-level problem size.

2) We demonstrate the solution technique effectiveness by solving a price-maker energy storage AC-OPF-
constrained market bidding problem. The results demonstrate higher achieved profits than with the DC market representation.

The paper is organized as follows. Section II provides mathematical foundation of the work and is divided in five subsections. Subsection II-A states the optimization problem, Subsection II-B describes the concept of fully connected neural networks, Subsection II-C explains the advantages, concept and our choice of hyperparameters of the used convolutional neural network, Subsection II-D describes the neural network training algorithm and Subsection II-E explains our iterative numerical scheme to solve the optimization problem at hand. The case study is presented in Section III with implementation details stated in Subsection III-A and test results in Subsection III-B. The final Section IV concludes the paper.

II. MATHEMATICAL MODELING

A. Optimization Model

In the following model we solve optimal ES bidding problem in the AC OPF network-constrained electricity market. The problem is of bilevel structure, i.e. the upper level maximizes the ES profit while the lower level maximizes social welfare due to supply and demand market bids. In the lower level we consider an exact nonconvex quadratic AC OPF formulation based on rectangular coordinates [37] notation, however, other notations such as polar [37] or current-voltage formulation based on rectangular coordinates [37] notation, are also applicable.

The upper-level problem consists of objective function (1.1), where \( \lambda_t \) is the electricity price in each hour indexed by \( t \), and \( q_t^{ES} \) is the average ES power during one hour, i.e. energy, at the interface. Constraint (1.2) models the ES (dis)charging process, i.e change in its state-of-energy SoE \( \Delta \) and \( \eta_t^{dis} \). Constraint (1.3) sets limits to the state-of-energy (SoE), with \( \hat{\text{SoE}} \) being the maximum value. Constraints (1.4) and (1.5) limit the ES (dis)charged energy to \( \bar{q}_t^{ch} \) for charging and \( \bar{q}_t^{dis} \) for discharging. Binary variable \( x^c_t \) enables simultaneous charging and discharging. Finally, equation (1.6) combines charging and discharging into a single variable \( q_t^{ES} \). Optimization variables are written in formulas in normal font and contained in the variables set \( \Xi \), while the parameters are written in bold font.

\[
\text{Maximize} \quad \sum_{t} q_t^{ES} \cdot \lambda_t \tag{1.1}
\]

\[
\text{SoE}_t = \text{SoE}_{t-1} + q_t^{ch} \cdot \eta_t^{ch} - q_t^{dis} / \eta_t^{dis}, \quad \forall t \tag{1.2}
\]

\[
0 \leq \text{SoE}_t \leq \bar{\text{SoE}}, \quad \forall t \tag{1.3}
\]

\[
0 \leq q_t^{ch} \leq \bar{q}_t^{ch}, \quad \forall t \tag{1.4}
\]

\[
0 \leq q_t^{dis} \leq \bar{q}_t^{dis} \cdot (1 - x_t^{ch}), \quad \forall t \tag{1.5}
\]

\[
q_t^{ES} = q_t^{ch} - q_t^{dis}, \quad \forall t \tag{1.6}
\]

The lower level is only textually explained and not written here since we are bypassing it completely. The rectangular AC OPF consists of the objective function, the bus power balance constraints, the power flow equations, the line apparent power limits, the bus voltage limits, the generator production limits and the reference bus constraints. Mathematically challenging are the power flow equations and the lower bus voltage limit constraints, which do not conform to the traditional single-level reduction technique as they are nonconvex. Moreover, there are two additional convex, but nonlinear parts of the formulation. The considered objective function has quadratic cost coefficients and line apparent power limit constraints are of second-order cone form. Broader insights of different AC OPF formulations can be found in tutorial works such as [37].

The concept of bypassing the lower level completely is based on the fact that the locational marginal prices \( \lambda_t \) are essentially a function of the upper-level \( q_t^{ES} \) variables, i.e. the objective can be expressed as \( F(q_t^{ES}, q_2^{ES}, \ldots, q_{|\tau|}^{ES}) \), where \( |\tau| \) is the cardinality of set \( \tau \) of time steps. However, \( \hat{F} \) can not be expressed explicitly, so we replace the objective function \( F \) with the approximation \( \hat{F} \) from (1.7). The approximating function \( \hat{F} \) is given as the feed-forward neural network, so it can be expressed explicitly in terms of elementary mathematical functions. Essentially, we are solving a single-level optimization meta-model that maximizes (1.7) subject to constraints (1.2)–(1.6).

\[
\text{Max} \quad \hat{F}(q_1^{ES}, q_2^{ES}, \ldots, q_{|\tau|}^{ES}) \tag{1.7}
\]

The problem belongs to the mixed-integer nonlinear optimization class due to the nonlinear neural network function \( \hat{F} \) and due to \( x_t^{ch} \) being binary variables.

B. Feed-forward fully connected neural networks

A feed-forward fully connected NN (see Figure 2) consists of \( K \) layers, where each layer consists of a number of neurons [39]. The first layer is referred to as the input layer, the last layer as the output layer, while the intermediate layers are called hidden layers. Neurons in each layer are connected only to the neurons in the neighboring layers. Feed-forward means that the data flows from the input layer to the output layer, strictly from one layer to the next one and in only one direction. Fully connected means that each neuron is connected to every neuron in the neighboring layers. Finally, each neuron in every hidden layer performs a nonlinear transformation on the data by applying the so-called activation function. More precisely, a feed-forward fully connected NN is a function \( F: \mathbb{R}^{N_1} \rightarrow \mathbb{R}^{N_K} \), where \( N_1 \) is the number of neurons in the input layer, and \( N_K \) is the number of neurons in the output

Fig. 2. Example of a fully connected neural network.
layer. Function $F$ is a composition of the alternating affine maps $A_k : \mathbb{R}^{N_{k-1}} \rightarrow \mathbb{R}^{N_k}$, $k = 2, \ldots, K$ and the element-wise nonlinear activation functions $N_k : \mathbb{R}^{N_k} \rightarrow \mathbb{R}^{N_k}$, $k = 2, \ldots, K - 1$, such that

$$F = A_K \circ N_{K-1} \circ A_{K-1} \circ \cdots \circ N_3 \circ A_3 \circ N_2 \circ A_2,$$

where $N_k$ is the number of neurons in $k$-th layer.

Affine map $A_k$ can be written in a matrix form as

$$z_k := A_k(z_{k-1}) = W_k z_{k-1} + b_k, \quad \forall k = 1, \ldots, K,$$

where weight matrices $W_k$ have dimensions $N_k \times N_{k-1}$ and bias vectors $b_k$ have dimensions $N_k$.

For the activation functions we element-wise use the Softplus function,

$$z_k^+ := N_k^+(z_k) = \frac{\ln(1 + \exp(\beta \cdot z_k^i))}{\beta}, \quad \forall i = 1, \ldots, N_k,$$

where $z_k^i$ is the $i$-th component of vector $z_k$ and $\beta$ is the hyperparameter of the Softplus function. Notice that for large values of $\beta$, Softplus uniformly converges to a rectified linear unit (ReLU) activation function (see Figure 3), which is given element-wise by

$$z_k^+ = \max(z_k^i, 0), \quad \forall k = 1, \ldots, K, \quad \forall i = 1, \ldots, N_k.$$

ReLU activation function is commonly used in recent NN applications. The reason why we decided to use Softplus will be become clear in Section II-E.

C. Convolutional neural networks

A CNN can be regarded as a sub-type of a feed forward NN. It is generally not fully connected, and a large number of weight and bias elements of matrices $W_k$ and vectors $b_k$ share the same values, as affine maps $A_k$ are defined using the operation of matrix convolution.

Figure 4 depicts a deep CNN, describing the exact NN topology used in approximating function $F$ of our problem. Besides the input and output layers, we have six additional hidden layers. Unlike in a general NN, each layer in our CNN is described using the layer length $L_k$ and the number of channels $C_k$. The number of neurons in each layer is given by $N_k = L_k \cdot C_k$ and neurons are grouped in $L_k$ groups of size $C_k$. In Figure 4 a single square depicts one group of neurons.

The exact number of neurons within each group (the number of channels $C_k$) is written inside each square. The number of groups in each layer is given by a number written just below each layer. For instance, the total number of neurons in the second layer is equal to $24 \cdot 32 = 768$.

To define affine map $A_k$, each layer in a CNN has an additional integer hyperparameter called the kernel size $S_k$. In Figure 4 only the first (input) layer and the last of the hidden layers have kernel sizes $S_1 = 1$ and $S_7 = 1$. All the other hidden layers have kernel size equal to 3. Notice that the kernel size is not applicable to the output layer, as the output layer only collects the output of the last hidden layer and is not applying any further affine maps. All kernels are depicted by a number of empty squares equal to the kernel size $S_k$. A downward arrow indicates that a kernel window is sliding over the layer in steps, performing a computation of the affine map. This means that the convolution operation can be in each step regarded as a smaller affine map $A_{\hat{k}}$ that is defined only between the neurons in the groups covered by the kernel window in layer $k - 1$ and the neurons in the single output group in layer $k$. In each step of the convolution operation on layer $k - 1$ we use the same map $A_{\hat{k}}$.

Each convolution layer has two additional integer hyperparameters called a stride and a padding size. The stride is the number of groups by which each kernel window moves in every step of the computing convolution operation. The first and second layers have the stride equal to 1 and the third to fifth layers have the stride equal to 2. This is the reason why the lengths $L_k$ of the fourth to sixth layer are decreasing by a factor of 2. For the sixth and seventh layers, the stride is not applicable as the convolution operation is trivially performed only in the single possible position. The padding controls whether the kernel window can slide over the side of the layer or not. If we let the kernel windows slide over the side of the layer, as for the second to fifth layer, the padding is equal to 1 and we substitute zeros for the input in the convolution operation in place of the non-existing data. For the first, sixth and seventh layer, the padding is equal to 0, which means we do not let the kernel window slide over the side of the layer.

In matrix representation $W_k$ of affine map $A_k$, lot of matrix components are equal to zero and lot of other non-zero matrix components share the same values. We actually have, for the kernel size equal to 1 the block diagonal matrix $W_k$, and for the kernel size equal to 3 the block tridiagonal matrix $W_k$, see Figure 5 Every block is of size $C_k \times C_{k-1}$. For the kernel size equal to 1, every block in the block diagonal matrix $W_k$ representing affine map $A_k$ is exactly the same block. For the kernel size equal to 3, every 3 vertical blocks in the block tridiagonal matrix representation are exactly the same blocks. The bias vector $b_k$ also has repeating components. Regardless on the kernel size, components of $b_k$ repeat every $C_k$ entries, which means each neuron group in a single layer shares the same biases.

Notice that a CNN, for the same number of neurons, typically has much lower number of parameters defining affine maps $A_k$, than a fully connected NN. For instance, in our case the number of parameters defining map $A_3$ is
datasets, using optimizer. The dataset is split into training and validation loss. Before the start of the first epoch of training, \( W \) of the loss function on the validation dataset are used to assess training and not for a gradient computation. Computed values of the gradients are computed using backpropagation \([40]\) and recursively in multiple epochs over the training dataset, where the block diagonal or tridiagonal structure of matrices \( W \).

As customary in NN training, the optimization of \( W_k \) and \( b_k \) is performed using a variant of a gradient descent optimizer. The dataset is split into training and validation datasets, using 80 : 20 percent split ratio. The optimization is done in multiple epochs over the training dataset, where gradients are computed using backpropagation \([40]\) and automatic differentiation \([41]\) of NN. The validation dataset is used only for evaluating the loss function after every epoch of training and not for a gradient computation. Computed values of the loss function on the validation dataset are used to assess numerical viability of the training process and to select the best trained network, having the lowest value of the validation loss. Before the start of the first epoch of training, \( W_k \) and \( b_k \) are initialized to random values.

More details about our exact training procedure, together with all optimizer and training hyperparameter values are given in Section III-A.

E. Meta-optimization numerical scheme

We devised an iterative numerical scheme for computing a sequence of CNN approximations \( F_i \) of the otherwise intractable objective function \( F \). In each iteration \( i \) we first generate dataset \( D_i \) of random sampled vectors \( q^{ES}_{t,i}, q^{ES}_{2,i}, \ldots, q^{ES}_{|\tau|,i} \). For each \( t \) in \( \tau \), values of \( q^{ES}_{t,i} \) are independently and uniformly random sampled from the interval centered around \( q^{ES,\text{rad}}_{t,i} \) of length at most \( 2q^{ES,\text{rad}}_{t,i} \), respecting the condition \( -q^{\text{dis}}_{t,i} \leq q^{ES}_{t,i} \leq q^{\text{ch}}_{t,i} \). In the first iteration we set \( q^{ES,\text{rad}}_{t,i} \) equal to zero and \( q^{ES,\text{rad}}_{t,i} \) such that it allows all permissible values of \( q^{ES}_{t,i} \), i.e. \(-q^{\text{dis}}_{t,i} \leq q^{ES}_{t,i} \leq q^{\text{ch}}_{t,i} \).

In practice, we noticed that our numerical scheme runs better if we introduce an additional small relative tolerance \( \epsilon > 0 \) on the dataset creation. Thus, for the maximum length of the sampling interval we actually use \( 2q^{ES,\text{rad}}_{t,i} (1 + \epsilon) \) and allow all permissible values of \( q^{ES}_{t,i} \) to be from the interval \(-q^{\text{dis}}_{t,i} (1 + \epsilon) \leq q^{ES}_{t,i} \leq q^{\text{ch}}_{t,i} (1 + \epsilon) \). The intuition behind introducing the tolerance is that our CNN would better approximate objective function \( F \) for parameter values \( q^{ES}_{t,i} \) near the boundary values \(-q^{\text{dis}}_{t,i} \) and \( q^{\text{ch}}_{t,i} \), if the dataset is allowed to include values \( q^{ES}_{t,i} \) a bit outside the interval \([-q^{\text{dis}}_{t,i}, q^{\text{ch}}_{t,i}] \).

Each element in dataset \( D_i \) is now a pair of random vector \((q^{ES}_{1,i}, q^{ES}_{2,i}, \ldots, q^{ES}_{|\tau|,i})\) and a value \( F(q^{ES}_{1,i}, q^{ES}_{2,i}, \ldots, q^{ES}_{|\tau|,i}) \) of the target objective function \([11]\), computed by solving a single instance of the lower-level problem.

Next, in each iteration we separately train a number of CNNs, \( F_{i,n} \) indexed by \( n \in \{1, \ldots, M\} \), to approximate function \( F \). For every additional training on the same dataset \( D_i \) we obtain a subtly different CNN, as the training process is intrinsically stochastic (random initialized network weights and random sampled stochastic gradient descent mini batches).

Now, as every trained CNN is a function \( F_{i,n} \), we symbolically insert function \( F_{i,n} \) into the upper-level problem objective function. For this we use our modeling environment’s, which is AMPL, defining variables feature. Defining variables are a type of variables that are substituted out, potentially in a nested way, by their declaration expression before reaching the solver. This results in solving a single-level optimization meta-model which maximizes...
\[
\begin{align*}
\max_{\Xi} & \quad \tilde{F}_{i,n} \left( q_{\text{ES}, t,i}^{\text{ES}}, q_{\text{ES}, t,i}^{\text{ES}}, \ldots, q_{\text{ES}, t,i}^{\text{ES}} \right) \\
\text{subject to constraints} \quad \text{(1.2)-(1.3)} \quad \text{and} \\
0 \leq q_{\text{ch}, t,i}^{\text{ES}} \leq q_{\text{ch}}^{\text{ES}}, \ x_t^{\text{ch}}, \quad \forall t \\
0 \leq q_{\text{dis}, t,i}^{\text{ES}} \leq q_{\text{dis}}^{\text{ES}}, \ (1 - x_t^{\text{ch}}), \quad \forall t \\
q_{\text{ES}, t,i}^{\text{ES}} - q_{\text{ch}, t,i}^{\text{ES}} \leq q_{\text{ES}, t,i}^{\text{ES}} = \frac{1}{M} \sum_{n=1}^{M} q_{\text{ES}, t,i}^{\text{ES}} \\nq_{\text{ES}, t,i}^{\text{ES}} - q_{\text{dis}, t,i}^{\text{ES}} \leq q_{\text{ES}, t,i}^{\text{ES}} \leq q_{\text{ES}, t,i}^{\text{ES}} + q_{\text{ES}, t,i}^{\text{ES}} + q_{\text{ES}, t,i}^{\text{ES}} \\
\end{align*}
\]

for every trained CNN indexed by \( n \). The additional constraint \((1.12)\) is used to respect that dataset \( D_t \) is created centered around \( q_{t,i}^{\text{ES, cnt}} \) using an interval length at most \( 2q_{t,i}^{\text{ES, rad}} \).

Notice, as we used the differentiable Softplus activation function in our CNN, and as a CNN is just a composition of affine maps and element-wise activation functions, that \( F_{i,n} \) are differentiable functions. In case of using a ReLU activation function, the resulting \( F_{i,n} \) would not be differentiable. We experimentally established that lower values of the Softplus hyperparameter \( \beta \) result in lower overall neural network approximation accuracy, and higher values give rise to solver instabilities in the single-level meta-model optimization step. We also tried to use ReLU instead of Softplus, but we were plagued with solver instabilities and slowdown. Using Softplus showed to be much more efficient.

For every trained CNN we now have the computed profit \( \mathcal{C}_{i,n} = \max_{\Xi} \tilde{F}_{i,n}(q_{t,i}^{\text{ES}}) \) and the computed optimal ES (dis)charged energy \( \left( q_{1,i,n}^{\text{ES}}, q_{2,i,n}^{\text{ES}}, \ldots, q_{\text{ES}, t,i}^{\text{ES}} \right) \). Now we can verify what are the actual profits obtained for the computed optimal ES (dis)charging schedule. This is done by optimizing the lower-level problem independently of the upper level with fixed ES (dis)charging schedule. The actual profit is determined as \( \mathcal{V}_{i,n} = \max - \sum_{t} q_{\text{ES}, t,i}^{\text{ES}, \lambda_t}, \) where \( \lambda_t \) is in this case the bus balance constraint marginal, computed by default by many interior point solvers.

Additionally, we compute mean optimal ES energy exchange quantities

\[
q_{\text{ES}, t,i}^{\text{rad}} = \frac{1}{M} \sum_{n=1}^{M} q_{\text{ES}, t,i}^{\text{ES}, n} \quad \forall t \in \tau
\]

and for the computed mean vector \( \left( q_{1,i,n}^{\text{rad}}, q_{2,i,n}^{\text{rad}}, \ldots, q_{\text{ES}, t,i}^{\text{rad}} \right) \) we again optimize the lower-level problem with fixed ES charging values to the obtained mean optimal vector, arriving at the mean actual profit \( \bar{V}_{i,n} = -\sum_{t} q_{\text{ES}, t,i}^{\text{ES}, \lambda_t} \). Considering the mean actual profit is justified, because averaging over optimal solutions of many different CNN models \( F_{i,n} \), each one approximating an intractable objective function \( F \), it can result in a better mean solution. By looking at the actual test results (i.a. Tables III and V) we see that this approach is justified in practice, i.e., in some iterations the mean actual profit \( \bar{V}_{i,n} \) can be higher than any of the actual profits \( \mathcal{V}_{i,n} \).

Finally, we have to choose values of \( q_{1,i+1}^{\text{ES, rad}} \) and \( q_{t,i+1}^{\text{ES, rad}} \) for a next iteration of the meta-optimization scheme. For \( q_{t,i+1}^{\text{ES, rad}} \) we either chose the optimal ES (dis)charging quantities \( \left( q_{1,i,n}^{\text{ES}, n}, q_{2,i,n}^{\text{ES}, n}, \ldots, q_{\text{ES}, t,i}^{\text{ES}, n} \right) \) from CNN that achieved the highest actual profit \( \mathcal{V}_{i,n} \), or in the case \( \bar{V}_{i,n} \) is the highest profit, we chose the mean optimal ES (dis)charging quantities \( \left( q_{1,i,n}^{\text{ES}, n}, q_{2,i,n}^{\text{ES}, n}, \ldots, q_{\text{ES}, t,i}^{\text{ES}, n} \right) \).

To choose \( q_{1,i+1}^{\text{ES, rad}} \), we first compute the maximum over all \( t \in \tau \) of the standard deviations of samples \( \left\{ q_{t,i,n}^{\text{ES}, n} : 1 \leq n \leq M \right\} \). More precisely, we compute

\[
\sigma_i := \max_{t \in \tau} \text{std} \left\{ q_{t,i,n}^{\text{ES}, n} : 1 \leq n \leq M \right\}.
\]

For the next iteration we take a smaller value between the current \( q_{t,i+1}^{\text{ES, rad}} \) and a new estimate,

\[
q_{t,i+1}^{\text{ES, rad}} := \min \{ q_{t,i}^{\text{ES, rad}}, \gamma \cdot \sigma_i \},
\]

for every \( t \), where \( \gamma \) is a hyperparameter. Notice that \( q_{t,i+1}^{\text{ES, rad}} \) does not depend on \( t \). It has the same value for every \( t \).

For the next iteration of our meta-optimization scheme, dataset \( D_{t+1} \) is created around \( q_{t,i+1}^{\text{ES, cnt}} \), which is the best optimal ES (dis)charging schedule computed in the current iteration, i.e., ES charging and discharging energy quantities that produce the highest profit. The dataset width, which is decided by \( q_{t,i+1}^{\text{ES, rad}} \), is influenced by how close together are optimal ES schedules predicted by different CNNs trained in the current iteration. In case different CNNs produce relatively close optima, the computed standard deviation \( \sigma_i \) is relatively small, and a next iteration dataset is going to be concentrated around a smaller neighborhood of \( q_{t,i+1}^{\text{ES, cnt}} \). On the contrary, in case different CNNs produce optima that are more apart, the computed standard deviation \( \sigma_i \) is relatively large, and a next iteration dataset is going to span over a bigger neighborhood of \( q_{t,i+1}^{\text{ES, rad}} \). Notice that \( q_{t,i}^{\text{ES, rad}} \) is non-increasing between iterations.

In the end, we have to prescribe a stopping criterion for our meta-optimization scheme. We choose to stop further iterations if there is no improvement in the actual profit of the current iteration compared to the previous one. We empirically conclude (see Section III-B) that the convergence of our meta-optimization scheme is achieved in few iterations (see Tables III, IV and V). An overview of the numerical optimization scheme is provided in Algorithm I.

### III. Case Study

#### A. Implementation details

For the dataset creation, each instance of the lower-level problem, one for every dataset entry, was solved using AMPL running KNITRO 12.3 solver. A single dataset entry consists of a 24-dimensional floating point vector \( \left( q_1^{\text{ES}}, q_2^{\text{ES}}, \ldots, q_{\text{ES}, t}^{\text{ES}} \right) \) as an input and a single floating point value as an output, which corresponds to the computed upper-level profit for given \( \left( q_1^{\text{ES}}, q_2^{\text{ES}}, \ldots, q_{\text{ES}, t}^{\text{ES}} \right) \) values (computed as \( -\sum_{t} q_{\text{ES}, t,i}^{\text{ES}, \lambda_t} \), where \( \lambda_t \) is the marginal of an active power bus balance constraint at the ES location). In total, we used \( 10^5 \) dataset entries and thus the same number of independent lower levels to be solved. To reduce the computation time, computations were carried out in parallel running on a dual Intel Xeon CPU computer system over a total of 40 physical cores.

We implemented our CNN depicted in Figure II in Python using PyTorch library [42]. The exact hyperparameters of our
Algorithm 1 Numerical optimization scheme

1: repeat
2: Generate a new random dataset (10^5 entries)
3: Evaluate LL response for the dataset
4: Train 60 NNs to approximate LL response
5: Optimize the ULs with inserted NNs into objective function
6: Determine actual profits by optimizing LL with fixed ES (dis)charging schedule
7: Select the best actual solution out of:
   - the best direct result;
   - the result obtained averaging decisions from all optimized NNs;
8: For the next iteration, reduce and concentrate the dataset spatial size in the neighborhood of the best solution found from this iteration
9: until The best solution is worse than in the preceding iteration

CNN are already described in detail in Section II-C. Regarding the remaining hyperparameter $\beta$ of the Softplus activation function, after conducting extensive training experiments, we settled on the value $\beta = 50$.

Training of the CNN was implemented using fast.ai library [43], which allowed us to easily implement several state-of-the-art optimizer features, likely improving over the nowadays rather standard Adam [44] and stochastic gradient descent optimizers.

Overall, the training procedure was very similar to the one employed in [45]. For the optimizer we used the Ranger algorithm [46], which can stabilize the start of the training by using the Rectified Adam (RAdam) optimizer [47]. Another feature we used is the parameter lookahead, which avoids overshooting a good local minima in parameter space [48], thus stabilizing the rest of the training process. The learning rate and weight decay were controlled during the training using the flat-cosine one-cycle policy. This produces convergence to a broader optimum, allowing better generalization for the trained neural network [49].

RAdam has several important hyperparameters that greatly influence the speed of the training convergence and the quality of the obtained trained neural network in respect to generalizability. These hyperparameters are: the number of epochs of training, the maximum learning rate, the training batch size, the weight decay factor and the exponential decay rates of the first and second moments. We experimented with training thousands of models in order to find the combination of the optimizer hyperparameters that work well for our CNN topology and dataset properties. Finally, we fixed the number of epochs to 500, maximum learning rate to 0.003, training batch size to 128, weight decay factor to 0.01 and exponential decay rates of the first and second moments to values 0.95 and 0.85.

In every iteration of the meta-optimization scheme we trained a total of $M = 60$ CNNs on the same dataset. Our computer system was equipped with 6 Nvidia Quadro RTX 6000 GPUs, each having 16 Gb of RAM. As the CNN and dataset sizes were relatively small and used only a fraction of our GPU memory and computational power, we could efficiently train all CNNs in parallel, training 10 CNNs per GPU. The slowdown induced by parallel training of 10 models per GPU is insignificant compared to alternatively much longer time of sequential single-model training.

In the meta-optimization scheme there are two hyperparameters to consider. After experimenting with different values, for the relative tolerance of the dataset creation we take $\epsilon = 0.1$, and for the other hyperparameter we take $\gamma = 5$. A value of hyperparameter $\gamma$ influences the decreasing rate of $q_{t,i,rad}^{ES}$ through subsequent iterations. Using lower values of $\gamma$ produces lower values of $q_{t,i,rad}^{ES}$ in later iterations, which can lead to a sub-optimal optimization result in the end. The chosen value of $\gamma$ is a compromise, in a sense that a higher value tends to slow down the speed of convergence. By using $\gamma = 5$, the reduction of $q_{t,i,rad}^{ES}$ is not overly aggressive. On the other end, it stabilizes our meta-optimization scheme, which leads to finding better optimums. Table I shows the $q_{t,i,rad}^{ES}$ decrease throughout all iterations from the case study. From the table we see that the radius decrease is significant, but not too sudden.

| Iter | 3_1mbd | 57_ieee | 73_ieee_rts |
|------|--------|---------|-------------|
| 1    | 0.6    | 0.6     | 0.6         |
| 2    | 0.0618 | 0.3265  | 0.4894      |
| 3    | 0.0319 | 0.2123  | 0.3465      |
| 4    | 0.0129 | 0.1876  | 0.3465      |
| 5    | 0.0129 |
| 6    | 0.0129 |

B. Test results

We tested our method on three separate transmission system meshed networks from PGLib-OPF [50] library: 3_1mbd, 57_ieee and 73_ieee_rts. A time dimension was added to the data by applying the load scaling factors for winter workdays available from IEEE RTS-96 [51]. Set of time steps $\tau$ has 24 elements for different hours in a single day. In case of 73_ieee_rts network, we have also applied 0.85 scaling factor to the transmission lines capacities to induce congestion. The networks were otherwise unmodified. For an ES to have an impact on the energy market prices, a feature for which the bilevel modeling is used for, it has to be very large. Thus, we model the ES with 100 MWh (1 p.u.) capacity. Charging and discharging efficiencies were both set to 90% and maximum ES (dis)charging power to 60 MW.

In Table I we present an average wall time per iteration and for different steps of our meta-optimization scheme. Dataset creation is a cumulative time for three sub-steps: generation of $10^5$ random vectors of $q_{t,i}^{ES}$, solving lower-level problems for every dataset entry, and data format post-processing of the generated dataset, which mostly include disk input-output (IO) operations. Most of the time is consumed for solving $10^5$ lower-level problems. Notice that bigger transmission system networks require more solver time. NN training is the time consumed for parallel CNN models training. Notice that this time does not depend on the transmission system...
network size, as it depends solely on the CNN and training hyperparameters. The last step is solving the meta-models, which is done sequentially for all 60 trained CNNs. A possible speedup of using parallel computations in this step would not be significant compared to the total time used per iteration. Notice that the time for solving the meta-models can vary greatly between different CNNs and different iterations. We consider this to be a normal solver behavior occurring due to binary variables \( x_i \). For the dataset creation and the NN training, the average wall time is pretty much unchanged between different iterations, so we supply only the mean times without the standard deviation.

In Tables II, IV and V we present optimization results in terms of the ES computed and actual profits acquired in three different transmission systems. Actual profits are obtained in the verification Step 6 of Algorithm II by optimizing the lower level with fixed ES charging decisions as explained in Section II-E. We also compare actual profits achieved by our method to actual profits achieved by solving a standard bilevel ES market optimal bidding using the DC OPF model in the lower level. Actual DC OPF profits are profits that would occur in the AC OPF market, but by using bidding decisions from the DC OPF bilevel model. In our tables, the best NN computed profit is the maximum value of \( C_{i,n} \) over all 60 NNs, the best NN actual profit is the maximum value of \( V_{i,n} \) over all 60 NNs, and the mean \( q^{ES}_{\text{actual}} \) profit is \( \bar{V}_n \), where \( i \) is the iteration number and the NN \( n \). By design, the best profit is always achieved in the penultimate iteration of our method, as a worse profit in the last iteration actually triggers the stopping criterion. The number of required iterations differs between the transmission systems and ranges from 4 to 6 iterations. Profit increase over the DC OPF model was 1.5% for 3_lmbd, 9.3% for 57_ieee, and 11.8% for 73_ieee network. From these tables it is observable that we also achieved a high first iteration accuracy, since the greatest second iteration improvement of the actual profit is only 0.03%. Notice that the total time per iteration presented is somewhat higher than a component total time in Table II as it includes an additional overhead for some data reformatting and IO disk operations. Also, notice that we decided to present profits using up to four decimal places, so that small improvements between subsequent iterations in Table II become visible, which also reaffirms the optimality of the first iteration result.

### IV. Conclusion

In this paper we present a novel numerical method, which utilizes deep convolutional neural networks to efficiently solve a wide class of bilevel optimization problems arising in deregulated power systems. Our method uses evolutionary meta-modeling to bypass the lower-level problem completely, thus it is insensitive to the lower-level complexity, which is the main culprit in rendering bilevel optimization problems intractable. We can successfully deal with nonlinear nonconvex lower-levels that include binary variables, as long as the lower-level can be efficiently solved as a single-level problem by treating all upper level variables as parameters. We used our method to specifically solve ES market participation problem using an AC OPF model in the lower level, which would enable the market operators to perform market clearing using the AC OPF instead of much less accurate DC OPF. However, the proposed framework is generally applicable for any other bilevel optimization problem in the power systems domain.

Additionally, our method is scalable in terms of the required precision versus the run time. Using larger training datasets we could train a neural network having more parameters than we used here. We would obtain more accurate optimums, but would also require longer run time. On the other hand, if we are satisfied with lower precision, we could use a smaller dataset for training a smaller neural network, and our method would run faster.

### REFERENCES

[1] IEEE, “IEEE Xplore”. Accessed: Oct. 4 2021. [Online]. Available: https://ieeexplore.ieee.org/Xplore/home.jsp.

[2] J. M. Arroyo and F. D. Galiana, “On the solution of the bilevel programming formulation of the terrorist threat problem,” IEEE Trans. Power Syst., vol. 20, no. 2, pp. 789–797, May 2005.

[3] I. Mombra, S. Wogrin and T. Gómez San Román, “Retail Pricing: A Bilevel Program for PEV Aggregator Decisions Using Indirect Load Control,” IEEE Trans. Power Syst., vol. 31, no. 1, pp. 464–473, Jan. 2016.
