Learning to Solve Network Flow Problems via Neural Decoding

Yize Chen and Baosen Zhang

Abstract

Many decision-making problems in engineering applications such as transportation, power system and operations research require repeatedly solving large-scale linear programming problems with a large number of different inputs. For example, in energy systems with high levels of uncertain renewable resources, tens of thousands of scenarios may need to be solved every few minutes. Standard iterative algorithms for linear network flow problems, even though highly efficient, becomes a bottleneck in these applications. In this work, we propose a novel learning approach to accelerate the solving process. By leveraging the rich theory and economic interpretations of LP duality, we interpret the output of the neural network as a noisy codeword, where the codebook is given by the optimization problem’s KKT conditions. We propose a feedforward decoding strategy that finds the optimal set of active constraints. This design is error correcting and can offer orders of magnitude speedup compared to current state-of-the-art iterative solvers, while providing much better solutions in terms of feasibility and optimality compared to end-to-end learning approaches.

I. INTRODUCTION

In many engineering applications, optimization programs are solved repeatedly to find real-time decisions. Among them, network flow problems form an important class and have been studied for decades with wide-ranging applications. They arise naturally in the context of transportation, networking, communication and energy systems. In many of these settings, a network flow problem takes the form of a linear program (LP) [1].

Despite being an LP with theoretical guarantees on performance and runtime, computational challenges still exist in practice. Often these challenges are due to the increased stochasticity and the low-latency requirements of real-time applications. The motivating application of this paper is power systems, where intermittent and random renewable resources are being integrated into the grid at all levels [2], [3]. These new resources can create new flow patterns and sudden generator ramps that fundamentally transform how systems operate. Inadequate planning for these events can lead to significant losses of welfare, as demonstrated by the recent rolling blackouts in Texas [4] and California [5].

The resource allocation process in power systems is called the optimal power flow (OPF) problem, where it takes the form of a network flow problem that minimizes the cost of power generation to satisfy the loads, subject to all of the physical network constraints (e.g., generator limits and line capacities) [6]. Typically, a linear version called DCOPF—an LP problem—is often applied in practice and solved periodically (e.g., every 5 minutes) to find the optimal operating conditions [7]. Because of the uncertainties brought on by the renewables on many of the nodes, the number of generation and load scenarios that need to be considered are starting to grow exponentially. Despite the inherent similarity between scenarios, the OPF problem needs to be resolved. Even if each one of the LPs takes 0.1 seconds using modern solvers, not all of them can be completed within the required time period. Therefore, using machine learning to learn the mapping between variable input load profiles and the corresponding optimal generation outputs has gained significant attention, since making inference via a trained architecture can offer orders of magnitude of computational speedup compared to an iterative solver [8], [9].

The learning-theoretic question becomes whether neural networks (NNs) can be used to directly make decisions in optimization problems, and in particular, whether it can learn the solutions of LPs as a function of the changes in the problem data. Surprisingly, the answer to this question has been largely negative. In [10], [11], convex optimization problems are embedded as layers to standard neural networks, with the rationale that directly learning the mapping from problem data to solutions is difficult, especially when learned solutions need to satisfy all constraints at the same time.

In this paper, we answer the question of whether a (standard) neural network can learn the solution of LP problems in the affirmative. Instead of viewing it as an end-to-end learning task, we leverage the rich algorithmic understanding of LPs as well as the economic interpretation of the primal and dual variables to offer a new solution architecture. Our workflow is shown in Figure 1. Concretely, we construct a neural network that takes the net load at each node as the input and the optimal system costs as the output. However, we do not take these outputs as the solution of the LP. Rather, using the neural network, we compute the gradient of the cost with respect to the net loads. Identifying these as the dual variables, we use them to predict the binding nodal and line constraints. Once these constraints are found, the optimal solutions are given by solving a simple linear system of equations. The overall procedure can be seen as an efficient surrogate learning model for optimization solvers. We make the following contributions in this work:

1) We provide a novel architecture to solve network flow problems that arise in many applications. Instead of using a neural network as an end-to-end model, we use it to learn the gradients of the mapping between the problem data and the

Y. Chen and B. Zhang are with the Department of Electrical and Computer Engineering at the University of Washington, emails: {yizechen, zhangbao}@uw.edu.
optimal cost. Interpreting these as dual variables, we provide extremely efficient ways to identify the active constraints in the problem.

2) Our method can be thought as decoding a system where the codebook is given by the KKT conditions. In particular, it is error correcting: even if output of the neural network is noisy, the correct set of active constraints at optimal solutions can still be identified.

3) We show that our method can provide more than an order of magnitude speedup compared to current state-of-the-art iterative solvers. It also performs much better than other end-to-end neural network approaches for LPs in terms of solution feasibility and optimality.

A. Related Work

Learning to Solve Optimization Our work falls under the category of using machine learning to solve optimization problems, which has a long history dating back to at least [12], [13]. A line of research considers using deep learning to solve combinatorial optimization problems typically by developing new heuristics [14], [15]. More similar to this paper is the line of work that tries to solve convex optimization problem by directly using supervised learning to find the optimal mapping from input data to the optimal solution, with applications in power systems [9], [16], scheduling in wireless communications [17], [18], and resource management [19]. These algorithms can be regarded as end-to-end behavior cloning of expert policies [20].

The challenge of using end-to-end models is that they can suffer from compounding errors and poor generalization performances, especially when hard constraints need to be satisfied. Our work is also related to [21], [22], where authors proposed to predict the set of active constraints at optimality. The intuition is that although the number of possible optimal active constraint sets are exponential in the size of the system, only a few of them are relevant for most input data. Yet such an approach is also purely based on expert policies restricted by training data, and learned neural networks has difficulties in making predictions on unseen inputs. In the cases of [3], [23], [24], the authors describe reinforcement learning formulation to model the interaction between learners’ decisions and rewards, but there is no guarantee the solutions satisfy all constraints. It is challenging to model constraint satisfaction as a reward signal, since small variations on input data can lead to large changes in the set of binding constraints.

Fast Solvers for Engineering Applications In many engineering and operations settings, optimization problems of specific structures are solved repeatedly with stringent latency requirements [25]. Neural networks, with good performances across many different domains, has been proposed as surrogates to replace iterative solvers. However, without utilizing the known structures of optimization model formulation as well as model parameters, the end-to-end model may be hard to generalize to unseen input instances, while the resulting solutions can violate constraints. In [10], a small LP problem with 3 variables is shown to be hard to learn via supervised learning. Our work is also related to [26], [11], [27], [28], where differentiable convex optimization layers provide convenient venues of designing end-to-end models, but they still use standard solvers at runtime and do not achieve our goal of speeding up computations in time/resource-constrained applications while satisfying all constraints.

II. Modeling and Learning Formulation

In this section, we provide the formulation of two network flow problems, describe our motivation for developing fast machine learning algorithm, and the relationship between optimal solutions and the active constraints.

A. Network with Independent Edge Flows

A common type of network flow problems arising in logistics and communication network concerns the distribution of goods from origins (e.g., a manufacturing plant) to destinations (e.g., consumers) [29]. Given a graph $G = (V, E)$, the sources, destinations, and intermediate points are collectively modeled as nodes in the set $V = \{1, \ldots, n\}$. The sources have certain amounts of goods and each of the sinks has some demand that needs to be satisfied. The goods need not be sent directly
from source to destination and may be routed through intermediary points reflecting distribution centers or routers. For each of the node, we let $x_i$ denote its production and let $l_i$ denote the load. Without loss of generality, we allow a node to have both positive $x_i$ and $l_i$, that is, a node can both produce and consume goods. Therefore $x$ and $\ell$ are both in $\mathbb{R}^n$. Each of the edges in the network carries some flow to facilitate the distribution of the goods and they are related to the nodes through a conservation equation: the sum of the flows into a node must equal to its net load. Algebraically, suppose there are $m$ edges, each with flow $f_j$. Then the flows are related linearly to $x$ and $\ell$ through an incidence matrix, $A$, where $x + Af = \ell$. We associate a positive cost $c_i$ with the $i$'th node, interpreted as its unit cost of producing the good. The standard network flow model then takes the following form to minimize the total costs of production:

$$\begin{align*}
\min_{x,f} & \quad c^T x \\
\text{s.t.} & \quad 0 \leq x \leq \bar{x} \\
& \quad -\bar{f} \leq f \leq \bar{f} \\
& \quad x + Af = \ell
\end{align*}$$

where $x$ and $f$ are the optimization variables with upper and lower bounds as shown in (1b) and (1c), respectively. Note, if a node is a pure sink, we can set $\bar{x}_i = 0$. The matrix $A \in \mathbb{R}^{n \times m}$ is the incidence matrix of the graph $G$, with the convention that flows into a node is positive.

Since (1) is an LP, standard optimization solvers such as CVX or CPLEX can be used to solve (1) in polynomial time. Here we note that there are more constraints ($\approx 3n + 2m$ equality and inequality constraints) than the number of optimization variables ($n + m$) in the above formulation, not every constraint can be binding. Finding the binding or active constraints is the key step of this paper.

B. Optimal Power Flow Problem

The optimal power flow (OPF) problem is one of the most important problems in power system operations. It is a class of a network flow problem, with a goal of minimizing generation costs while satisfying all the loads and flow constraints. The key difference with the problem in (1) is that the flows in a power system cannot be set independently. Namely, they must obey Kirchoff’s voltage laws, which states that a linear combination of the flows must sum up to zero in a cycle in the network. For example, in Fig. 2 for the standard network on the left, the flow on each link can be changed independently to each other. On the other hand, for the power system on the right of Fig. 2 (a linear combination of) flows are constrained to sum to 0. More modeling details can be found in Appendix A.

Because of these cycle constraints, the edge flows must lie in a subspace in $\mathbb{R}^m$. This space is the so-called fundamental flow space, with a dimension of $n - 1$. The rest of the $m - n + 1$ flows are uniquely determined by the cycle constraints. By selecting a spanning tree of the network where flows on the edges in the tree are fundamental, the rest of the flows on the edges are determined through the cycle constraints. Algebraically, we use $f$ to denote the flows in a cycle basis and a matrix $K \in \mathbb{R}^{m \times (n-1)}$ to map it to all of the flows. Then the optimization problem becomes:

$$\begin{align*}
\min_{x,f} & \quad c^T x \\
\text{s.t.} & \quad 0 \leq x \leq \bar{x} \\
& \quad -\bar{f} \leq Kf \leq \bar{f} \\
& \quad x + \tilde{A}f = \ell,
\end{align*}$$

where $\tilde{A}$ is the modified incidence matrix mapping the fundamental flows to the nodal injections.
The OPF model is also an LP. However, as it becomes necessary to resolve for a large set of scenarios to accommodate renewable energy, we need to explore a faster surrogate algorithm.

C. Active Constraints

At the optimal solution of an LP, a constraint is called active if it is an equality constraint or an inequality constraint that binds. In this paper, we assume the LP is not degenerate, and the key result (Theorem 1) we use from linear programming is that the number of active constraints is exactly the same as the number of decision variables in the problem.

**Theorem 1.** For a nondegenerate linear programming problem with optimization variables, the optimal solution is determined by linearly independent active constraints.

The proof of the theorem is standard and found in many textbooks on linear programming [1], [31]. Conversely, if we know which constraints are active at the optimal, we can simply solve linear equations to find the optimal solution instead of an optimization problem. This is especially advantageous for network flow problem, of which there are very efficient equation solving algorithm with complexity almost linear in the number of variables [32], [33].

In this paper, we develop a novel machine learning algorithm to learn the set of active constraints without solving the LP. Notably, we do not attempt to directly learn the set of active constraints via a classification problem. Since there can be exponential number of combination of active constraints, this approach has proven to be difficult [22].

In this section, we present learning the active constraints as a decoding process, where the code is learned by a novel neural network implicitly. By interpreting the KKT conditions as a codebook, we demonstrate how to utilize the structure and parameter knowledge of the optimization model to train a neural network. Analysis in this section can also be extended to more general convex optimization problems.

Throughout this section, we work with the optimal power flow problem in (2), since the standard network flow problem in can be thought as a special case by taking to be identity and the cycle basis to include flows on all of the edges. Case of line flow costs can be analyzed similarly.

A. Derivatives of the LP

We can view the LP in (2) as a mapping from the load vectors to the cost of the objective function. We denote this function as with optimal value , which implicitly encodes the feasibility and optimality conditions. Since the output of is a scalar and is continuous in , it is much easier to learn than the vector of optimal solutions or the active constraints. Of course, we are rarely satisfied about knowing only the optimal cost. But this function has rich structure that can be exploited using the following theorem:

**Theorem 2.** Let be the optimal dual variables associated with equality constraints (2d). If for a given is feasible, then

\[ \nabla_\ell J^* = \mu^*, \]  \hspace{1cm} (3)

where \( \mu^*_i \) is bounded below by 0. Furthermore, the optimal solution for (2) is associated with the following active/inactive constraints:

\[ x_i = \begin{cases} \bar{x}_i, & \text{if } \mu^*_i - c_i > 0 \\ 0, & \text{if } \mu^*_i - c_i < 0 \\ (0, \bar{x}_i), & \text{otherwise} \end{cases} \]  \hspace{1cm} (4)

**Proof.** The proof follows from standard duality theory. We require feasibility of the input to rule out the dual being unbounded. Assuming that (2) is feasible, the Lagrangian is given by

\[ L(\ell, x, f, \mu, \hat{\lambda}, \nu, \lambda) = c^T x + \mu^T (\ell - x - \hat{A}f) + \lambda^T (Kf - \tilde{f}) - \hat{\lambda}^T (Kf + \tilde{f}) - \nu^T x + \tilde{\nu}^T (x - \bar{x}) \]  \hspace{1cm} (5)

where and are the dual variables associated with the capacity constraints in (2c), and is the dual variable associated with the equality constraint (2d). The dual variables and are nonnegative.

If , , , , , , are the optimal primal and dual solutions, then by strong duality

\[ J^*(\ell) = L(\ell, x^*, f^*, \mu^*, \hat{\lambda}^*, \tilde{\nu}^*, \lambda^*) \]

and differentiating (5) gives (3).
We adopt an economics argument here for the rest of the theorem and a rigorous proof through the KKT conditions is given in the Appendix [A]. Since \( \mu_i^* = \frac{\partial f}{\partial x_i} \), we can interpret it as the marginal cost of producing one more unit of good at node \( i \). Then \( \mu_i^* \geq 0 \) since all the cost are positive. If \( \mu_i^* < c_i \), the marginal cost of production is lower than the cost at node \( i \), which means that node \( i \) has zero production \( (x_i^* = 0) \). Conversely, if \( \mu_i^* > c_i \), then node \( i \) cannot produce more to satisfy the demand at a lower cost, then it must be at its upper bound \( \bar{x}_i \).

Theorem 2 states if \( \mu^* \) is known, then all active constraints at each of the nodes can be readily “decoded” by exploiting the parameters (cost vector and network topology) of original optimization problem. Of course, two question remains: 1) how do we find the active flow constraints and 2) how do we find \( \mu^* \)?

B. Finding Active Flow Constraints

Again we suppose that \( \mu^* \) is known (i.e., as the gradient of a learned neural network) and show how to find the active constraints of the edge flows in (2c). Continuing with the Lagrangian in (5), we can find the dual of (2):

\[
\begin{align*}
\max_{\mu, \lambda, \nu, \bar{\nu}} & \quad \mu^T \ell - \lambda^T \bar{\ell} - \bar{\nu}^T \bar{x} \\
\text{s.t.} & \quad c - \mu - \nu + \bar{\nu} = 0 \quad \text{(6a)} \\
& \quad -\bar{\lambda}^T \mu - K^T \lambda + \bar{\lambda}^T \bar{\lambda} = 0 \quad \text{(6b)} \\
& \quad \nu \geq 0, \bar{\nu} \geq 0, \lambda \geq 0, \bar{\lambda} \geq 0 \quad \text{(6c)} \\
& \quad |\mu| = 0, |\lambda| = 0, |\bar{\lambda}| = 0 \quad \text{(6d)}
\end{align*}
\]

If \( \mu^* \) is given, the dual variables associated with the nodal constraints and the edge flow constraints decouple, where the latter is:

\[
\begin{align*}
\max_{\lambda, \bar{\lambda}} & \quad -\lambda^T \bar{\ell} - \bar{\lambda}^T \bar{\ell} \\
\text{s.t.} & \quad K^T (\bar{\lambda} - \lambda) = \bar{\lambda}^T \mu^* \quad \text{(7a)} \\
& \quad \bar{\lambda} \geq 0, \lambda \geq 0 \quad \text{(7b)}
\end{align*}
\]

We can not directly find line constraint binding conditions due to the coupling between \( \bar{\lambda}, \lambda \) and \( \mu^* \). The following simple lemma is a useful way to rewrite (7).

**Lemma 1.** Assume that \( \bar{f} = f \) (symmetric edge capacities). Then the problem (7) is equivalent to

\[
\begin{align*}
\min_{\nu} & \quad ||\nu||_1 \\
\text{s.t.} & \quad \bar{K}^T (\nu) = \bar{\lambda}^T \mu^* \quad \text{(8a)} \\
& \quad |\nu_i| = |\bar{f}_i \cdot \bar{\lambda}_i - \bar{f}_i \cdot \lambda_i|, \text{ and } \bar{K} = \text{diag}(1/\bar{f}_1, \ldots, 1/\bar{f}_m)K.
\end{align*}
\]

For the proof see Appendix [B]. The assumption that the edge capacity is symmetric holds for most undirected networks seen in practice.

For the standard network flow problem in (1), the matrix \( K \) is the identity and (8b) is a full rank linear equation. Therefore, the optimal \( \nu \) (and hence the multipliers) can be found by inspection. However, for the OPF problem in (2), the cycle constraints make solving (8) less trivial.

Fortunately, transformed \( \ell_1 \) minimization problem in (8) is extremely well studied, since it falls exactly into the regime of sparse signal recovery in compressed sensing, where signal \( \nu \) needs to be recovered via observation \( \bar{K}^T \mu^* \). Note that in sparse recovery, \( \ell_1 \) minimization is a surrogate problem where the ultimate goal is to find the sparsest solution to the problem. For us, (8) is the exact problem we want to solve to finish the active constraints identification.

Because there are limited number of active constraints which is equivalent to limited number of nonzero entries in \( \nu \), while \( K \) has most of entries equal to 0 which allows for fast matrix multiplication, there are many algorithms that can solve (8) extremely efficiently. For example, a family of fast iterative greedy algorithm such as iterative hard thresholding (IHT) can be used to find \( \nu \) [34] in a few iterations.

This section completes our decoding scheme, where KKT conditions and dual problems are first utilized to decode a set of dual variables as the status of inequality constraints, while the remaining constraints can be identified by a smaller, subsequent \( \ell_1 \) minimization step.
There are 5 optimization variables (3 generators, 2 line flows), 6 inequality constraints associated with generators and 6 inequality constraints associated with line flows. We generate trained neural network can be used for optimal solution predictions for fast implementation.

Analyses in the previous sections assumed that the optimal dual variable $\mu^*$ is available. We now design a neural network to learn $\mu^*$ by following Theorem 2. The training is implemented offline using supervised data by collecting solution data of LP with different $\ell$. We use the optimal cost, optimal multiplier (readily available from most solvers) and the active constraints set as the labeled data.

We do not directly use a neural network to learn $\mu^*$. Rather, We fit a neural network (parameterized by $\theta$) $g_\theta(\ell)$ that maps optimization model input $\ell$ to optimal costs. Let $h(g_\theta(\ell))$ denote the binary vector indicating the active constraints determined by learner’s solution based on the procedure described in the last section. Interestingly, the structure and parameters of the underlying optimization problem provides us with a number of terms in the loss function:

- Regression loss defined between $g_\theta(\ell)$ and $J^*(\ell)$ over the neural network’s output;
- Regression loss for optimal dual variable defined between $\nabla_\ell g_\theta(\ell)$ and $\mu^*$;
- Distance loss defined between $h(g_\theta(\ell))$ and $s^*(\ell)$, where $s^*(\cdot)$ is the binary vector indicating active constraints at the optimal solution.

These three terms sum up to our model’s training loss

$$
L(\theta) = \|g_\theta(\ell) - J^*(\ell)\|_2^2 + \gamma_1 \|\nabla_\ell g_\theta(\ell) - \mu^*\|_2^2 + \gamma_2 \|h(g_\theta(\ell)) - s^*(\ell)\|_H
$$

where $\|h(g_\theta(\ell)) - s^*(\ell)\|_H$ is the Hamming distance between active constraint set and $\gamma_1, \gamma_2$ are penalty parameters. The trained neural network can be used for optimal solution predictions for fast implementation.

In Fig 3, we show a toy example of proposed learning approach on a 3-bus OPF example with $c_1 = 1.0, 1.5, 2.4$ respectively. There are 5 optimization variables (3 generators, 2 line flows), 6 inequality constraints associated with generators and 6 inequality constraints associated with line flows. We generate 100 test load samples which are generated from a uniform distribution, and visualize $\mu^*$. Our learning approach accurately predicts $\mu^*$, and is then able to identify binding generation constraints. We also notice that with 100 different $\ell$, there are only 3 different $\mu^*$, which shows that the pattern of active constraints are more structured compared to $\ell$ and $x^*$.

V. Robustness to Errors

The discussion in Section III motivates us to use a neural network to learn the optimal costs along with a “codeword” associated with dual variables. Because we are dealing with continuous values, the prediction of the neural network will invariably have errors. Therefore, it is important that the errors made do not add up and cause incorrect identification of the active constraints. Using an analogy from communication theory, we think of the derivatives of the learned neural network $g_\theta(\ell)$ as noisy versions of a codeword, and provide an error-correcting approach to decode active constraints that is robust to errors made by the neural network. In addition, this approach leads to a way to use dictionary learning to further speed up the solution process.

A. Error Correction

Observe that for a given $\mu^*$ and some noise $\delta$, though the optimal solution for the optimization problem may be different for $\mu^*$ and $\mu^* + \delta$, the set of active constraints at optimal solutions can remain the same. Therefore, there exists a region

1For binary vector $v$, Hamming norm $\|v\|_H$ is defined as the number of non-zero entries of vector $v$. 
Algorithm 1 Neural Decoder for Active Constraints

Input: $\ell$, trained model $g_\theta$, number of active constraints $K$
Input: (Optional) Dictionary $\{ \tilde{A}^T \mu_{(k)} \}$, $\tilde{A}^T \mu_{(k)}$, $f_{(k)}$

Parameters: $\epsilon > 0$, Optimization model $\tilde{A}, \tilde{f}, c$, $FLAG = 0$

1: Find $\nabla_{\ell} g_\theta(\ell)$
2: for $i = 1, \ldots, n$ do
3: if $\nabla_{l_i} g_\theta(\ell) - c_i < -\epsilon$ then
4: $x_i = 0$, $K = K - 1$
5: else if $\nabla_{l_i} g_\theta(\ell) - c_i > \epsilon$ then
6: $x_i = \bar{x}_i$, $K = K - 1$
7: end if
8: end for
9: for $j = 1, \ldots, |M|$ do
10: $\delta(j) = \tilde{A} \nabla_{l_j} g_\theta(\ell) - \tilde{A}^T \mu_{(j)}$
11: if $\delta(j) \in \tilde{P} \tilde{A}^T \mu_{(j)}$ then
12: Identify active flow constraints $f_{(j)}$, $FLAG = 1$
13: break
14: end if
15: end for
16: if $FLAG = 0$ then
17: IHTSolve(8, sparsity=$K$)
18: end if
19: EquationSolve($x + Af = \ell$, active constraints set)

around a ground truth optimal dual solution where as long as the noise does not push the solution outside of this region, the set of active constraints remains the same.

It turns out these regions are polytopes and easily characterized by solving another linear program. This falls under the well studied area of linear programming sensitivity analysis [1], [35]. The next lemma helps us to utilize these results to obtain an estimate of the testing error based on the training error of the neural network:

**Lemma 2.** Consider a given $\ell$ and its associated optimal dual variables $\mu^*$. There exists a polytope $\tilde{P} \mu^*$ around $\mu^*$ such that if $\tilde{\mu} - \mu^* \in \tilde{P} \mu^*$, then the active constraints determined by the procedure in Algorithm 1 given $\tilde{\mu}$ is correct. The set $\tilde{P} \mu^*$ is computable by a linear program.

The proof of this lemma is given in the Appendix [C]. It shows that our approach is robust to noise, since we do not require that the value of predictions $\nabla_{\ell} g_\theta(\ell)$ to be exact, and there is no compounding of errors as compared to other end-to-end approaches. It also provides a way to check whether training is good enough: we can compare the error between the learned $\mu$ and the actual $\mu^*$, and if it falls within the polytope $\tilde{P} \mu^*$, then we are confident that the training results would be accurate.

![Fig. 4](image-url) The optimal active line constraints prediction results for Network Flow problem on different size of graphs. For varying problem input distributions (low, medium and high variances), our proposed Neural Decoder approach can learn the optimal set of active constraints with lowest infeasibility percentage comparing to active set classification and end-to-end prediction of line flow values.
with high probability of feasibility and little optimality gap compared to optimal solutions. We can directly identify the value of \( \lambda \) to generate the training set, we use CVXPY \[37\] powered by a CVXOPT solver \[38\] to solve (1) and (2). For each network, very competitive results not only in achieving accurate results in terms of predicting \( J \), due to the fact that it is hard to generalize to unseen \( \ell \) in all test networks and input distributions. Within all the feasible test instances, the mean solution costs compared to optimal solutions provided by CVXPY is within 0\%.

In Fig. 4 we show the probability of encountering infeasible solutions by using three methods \( \ell \) as the input for all three methods. Once we get the prediction results on active constraints from \( \ell \) and \( \mu^* \), we also train and evaluate two other learning models: 1) solution quality of the data as test samples. Overview of evaluated tasks is listed in Table I.

| Graph Problem | Nodal Constraints | Line Constraints | Problem Input |
|---------------|-------------------|------------------|--------------|
| Network Flow  | up to 200         | up to 598        | nodal net demands |
| OPF IEEE 14-Bus | up to 24         | up to 47         | nodal loads |
| OPF IEEE 39-Bus | up to 59         | up to 100        | nodal loads |

**TABLE I**

**Experiments Overview.**

**VI. Experimental Results**

In this section, we show simulation results of the proposed learning approach (Neural Decoder) on a set of network flow optimization tasks, and demonstrate improvements over other learning alternatives. Specifically, we examine 1) solution quality in terms of constraint satisfaction and optimality; 2) performance under different problem input settings; and 3) computational efficiency over existing convex optimization solver. Simulations are run on an unloaded Macbook Pro with Intel Core i5 8259U CPU @ 2.30GHz.

**Training Set Generation.** We evaluate our approach on 3 different size of network flow problems \([1]\) consisting 20, 50, 200 nodes with randomly generated connectivity, and 2 power system benchmarks: the IEEE 14-Bus and IEEE 39-Bus system \([50]\). To generate the training set, we use CVXPY \([37]\) powered by a CVXOPT solver \([38]\) to solve (1) and (2). For each network model, we generate \( \ell \) by sampling from uniform distribution. In the network flow instances, the largest deviation from the nominal net load vector is 30\%, while in OPF instances, the largest deviation from nodal loads is 80\%. We solve 60,000 data samples using CVXPY for each network model under each variance setting. We also calculate the mean computation time for solving each data sample. We split 20\% of the data as test samples. Overview of evaluated tasks is listed in Table I.

**Baselines** In order to evaluate whether proposed learning algorithm design has a better performance in terms of predicting optimal solutions that satisfy all constraints, we also train and evaluate two other learning models:

**End to End prediction:** Following \([9]\), we construct a 4 layer neural network to fit the regression task of predicting optimal solution based on input \( \ell \). Standard mean squared error is adopted as training loss.

**Classification for active constraint sets:** Following \([22]\), we construct a 4 layer neural network to predict the set of active constraints at optimal solution. We use one-hot encoding for different set of active constraints, and use cross-entropy as the loss function.

For the neural decoder model, the largest model we use is a 3 layer fully-connected neural network for the 39-bus OPF task and 200-node network flow model. For other tasks, we find that even a 2 layer neural network with 50 neurons can produce very competitive results not only in achieving accurate results in terms of predicting \( J^*(\ell) \), but also in finding decision values with high probability of feasibility and little optimality gap compared to optimal solutions.

We use standard linear equation solver to find final solutions. More simulation setup details can be found in Appendix I.

**Experimental Results** In Fig. 4 we show the probability of encountering infeasible solutions by using three methods on testing data of network flow problem \([1]\). Infeasible solutions occur either when there is nodal imbalance \( \ell \) or solved line flow is out of line capacity bound. Proposed Neural Decoder can generalize well to larger networks, and keeps the lowest infeasibility in all test networks and input distributions. Within all the feasible test instances, the mean solution costs compared to optimal solutions provided by CVXPY is within 0.5\%. With more uncertainties on load input distributions, all three methods predict more infeasible solutions. The end-to-end prediction is hard to generalize to larger network and larger variances, partly due to the fact that it is hard to generalize to unseen \( \ell \) when there is growing number of constraints. The classification approach is also not a proper learning strategy for larger-scale optimization problems, since the growing number of possible combination

---

2We allow a 5\% mismatch when evaluating each node’s equality constraint.
of active constraints leads to huge one-hot encodings at the classifier’s output. What is more, for unseen combination of active constraints, the neural network classifier is not able to predict the active constraints correctly.

Results on learning for OPF problem are listed in Table II where we report the accuracy of active generators’ constraints and lines’ constraints separately. For all the feasible solutions provided by Neural Decoder on test samples, the mean cost increase compared to solvers’ solution is within 0.7%. We notice that proposed method can provide solutions with lower percentage of infeasibility than the sum of error on active generator and line constraints predictions. This is because identification steps of active generators and active lines are performed in sequence and we are making use of the information on total number of active constraints, so the error on each step does not add and lead to infeasible solutions. On the contrary, end to end prediction and classification produce many more infeasible solutions.

In Figure 5 we visualize the training progress during 50 training epochs on the OPF task for IEEE 39-bus system. We also evaluate model predictions’ feasibility on test dataset after each epoch. Both the regression loss for end to end training and Neural Decoder training converges quickly to loss values smaller than 0.01. However, after the few epochs’ progress on increasing the solution feasibility on testing data, the end-to-end predictions fail to provide practical, feasible solutions. Solution feasibility of end-to-end approach also varies significantly for different epochs, indicating that the mean squared error defined on solution prediction is not a good indicator for solution feasibility.

Compared to the solving process of CVXOPT, proposed neural decoder provides an order-of-magnitude speed-up in all testing
benchmarks as shown in Table [III]. We note that we are using standard Python packages for neural network derivations and equation solvers, while further acceleration can be achieved for the neural decoder. For example, taking a batch of evaluating samples to calculate \( \nabla E_{\theta}(\ell) \) together is faster than one-by-one derivation calculation. Special linear equation solver can be employed to solve the resulting linear equations once all active constraints are identified.

VII. CONCLUSION AND DISCUSSION

In this work we presented a unified learning framework for solving a family of network flow problems. Using neural networks and their derivatives, we incorporate the structure and parameters of the optimization model into the task of predicting active constraints. The method is error-correcting and achieves orders of magnitude of computation speed acceleration compared current state-of-the-art LP solvers. It also provides significant higher quality solutions in terms of optimality and constraint satisfaction compared to other end-to-end methods. Such fast and robust learning framework can be extended in several future research directions, such as learning to solve nonlinear convex optimization problems of specific structures, as well as solving robust and online optimization problems.

REFERENCES

[1] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*. Athena Scientific Belmont, MA, 1997, vol. 6.
[2] Z. Wen, D. O’Neill, and H. Maei, “Optimal demand response using device-based reinforcement learning,” *IEEE Transactions on Smart Grid*, vol. 6, no. 5, pp. 2312–2324, 2015.
[3] G. Dalal, E. Gilboa, and S. Mannor, “Hierarchical decision making in electricity grid management,” in *International Conference on Machine Learning*, 2016, pp. 2197–2206.
[4] J. Jimenez, “How the Texas power grid braces against rolling blackouts as summer heat looms,” *The Dallas Morning News*, 2019.
[5] I. Penn, “This did not go well: Inside PG&E’s blackout control room,” *The New York Times*, 2019.
[6] J. D. Glover, M. S. Sarma, and T. Overbye, *Power system analysis & design, SI version*. Cengage Learning, 2012.
[7] B. Stott, J. Jardim, and O. Alsaq, “De power flow revisited,” *IEEE Transactions on Power Systems*, vol. 24, no. 3, pp. 1290–1300, 2009.
[8] W. Deng, Y. Ji, and L. Tong, “Probabilistic forecasting and simulation of electricity markets via online dictionary learning,” arXiv preprint arXiv:1606.07855, 2016.
[9] X. Pan, T. Zhao, and M. Chen, “Deepopf: A deep neural network approach for security-constrained dc optimal power flow,” arXiv preprint arXiv:1910.14448, 2019.
[10] B. Amos and J. Z. Kolter, “Optnet: Differentiable optimization as a layer in neural networks,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 136–145.
[11] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, “Differentiable convex optimization layers,” in *Advances in Neural Information Processing Systems*, 2019, pp. 9558–9570.
[12] M. P. Kennedy and L. O. Chua, “Neural networks for nonlinear programming,” *IEEE Transactions on Circuits and Systems*, vol. 35, no. 5, pp. 554–562, 1988.
[13] W. E. Lillo, M. H. Loh, S. Hui, and S. H. Zak, “On solving constrained optimization problems with neural networks: A penalty method approach,” *IEEE Transactions on neural networks*, vol. 4, no. 6, pp. 931–940, 1993.
[14] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, “Learning combinatorial optimization algorithms over graphs,” in *Advances in Neural Information Processing Systems*, 2017, pp. 6348–6358.
[15] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi, “Exact combinatorial optimization with graph convolutional neural networks,” in *Advances in Neural Information Processing Systems*, 2019, pp. 15554–15566.
[16] R. Canyassse, G. Dalal, and S. Mannor, “Supervised learning for optimal power flow as a real-time proxy,” in *2017 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*. IEEE, 2017, pp. 1–5.
[17] X. Cao, R. Ma, L. Liu, H. Shi, Y. Cheng, and C. Sun, “A machine learning-based algorithm for joint scheduling and power control in wireless networks,” *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4308–4318, 2018.
[18] W. Cui, K. Shen, and W. Yu, “Spatial deep learning for deep feeding,” *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1248–1261, 2019.
[19] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, “Learning to optimize: Training deep neural networks for wireless resource management,” in *2017 IEEE 18th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2017, pp. 1–6.
[20] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Müller, J. Zhang et al., “End to end learning for self-driving cars,” arXiv preprint arXiv:1604.07316, 2016.
[21] D. Deka and S. Misra, “Learning for dc-opf: Classifying active sets using neural nets,” in *2019 IEEE Milan PowerTech*. IEEE, 2019, pp. 1–6.
[22] Y. Ng, S. Misra, L. A. Roald, and S. Backhaus, “Statistical learning for dc optimal power flow,” in *2018 Power Systems Computation Conference (PSCC)*. IEEE, 2018, pp. 1–7.
[23] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, “Resource management with deep reinforcement learning,” in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. IEEE, 2016.
[24] J. Song, R. Lanka, A. Zhao, Y. Yue, and M. Ono, “Learning to search via retrospective imitation,” arXiv preprint arXiv:1804.00846, 2018.
[25] S. S. Rao, *Introduction to linear optimization*. Athena Scientific Belmont, MA, 1997, vol. 6.
[26] J. L. Gross and J. Yellen, *Graph theory and its applications*. CRC press, 2005.
[27] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.
[28] Y. T. Lee and A. Sidford, “Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems,” in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, 2013, pp. 147–156.
[29] J. Savarino, “Convexity, polyhedrality, and undirected multicommodity flow,” in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 452460. [Online]. Available: https://doi.org/10.1145/3055399.3055501
[34] T. Blumensath and M. E. Davies, “Iterative thresholding for sparse approximations,” *Journal of Fourier analysis and Applications*, vol. 14, no. 5-6, pp. 629–654, 2008.

[35] B. Jansen, J. De Jong, C. Roos, and T. Terlaky, “Sensitivity analysis in linear programming: just be careful!” *European Journal of Operational Research*, vol. 101, no. 1, pp. 15–28, 1997.

[36] T. Athay, R. Podmore, and S. Virmani, “A practical method for the direct analysis of transient stability,” *IEEE Transactions on Power Apparatus and Systems*, no. 2, pp. 573–584, 1979.

[37] S. Diamond and S. Boyd, “Cvxpy: A python-embedded modeling language for convex optimization,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2909–2913, 2016.

[38] M. S. Andersen, J. Dahl, and L. Vandenberghe, “Cvxopt: A python package for convex optimization, version 1.1. 6.” Available at cvxopt.org, vol. 54, 2013.
respectively. Optimization involving $\bar{\lambda} = \bar{\nu} \odot \bar{f} - \bar{f}$ can be reformulated as

$$\min_{\nu, \bar{\nu}} \bar{\nu}^T \bar{x} \quad \text{(10a)}$$

subject to

$$\bar{\nu} - \nu = \mu^* - c, \quad \text{(10b)}$$

$$\bar{\nu} \geq 0, \nu \geq 0 \quad \text{(10c)}$$

which can read out the result without solving explicitly. Given that $\bar{\nu}, \nu \geq 0$, if $\mu_i^* - c > 0$, $\bar{\nu}_i > 0$, $\nu_i > 0$, and $x_i = \bar{x}_i$; if $\mu_i^* - c > 0$, $\nu_i = 0$, $\nu_i > 0$, and $x_i = 0$; if $\mu_i^* - c = 0$, $\bar{\nu}_i = \nu_i = 0$ and corresponding $x_i$ is not binding.

### B. Proof of Lemma 7

**Proof.** The original optimization problem is reproduced below:

$$\max_{\bar{\lambda}, \bar{\Delta}} -\bar{\lambda}^T \bar{f} - \bar{\lambda}^T \bar{f} \quad \text{(12a)}$$

subject to

$$K^T(\bar{\lambda} - \bar{\Delta}) = \bar{\Delta}^T \mu^* \quad \text{(12b)}$$

$$\bar{\lambda} \geq 0, \bar{\Delta} \geq 0. \quad \text{(12c)}$$

Assuming $\bar{f} = \bar{f}$ and let $\nu = \bar{\nu} \odot \bar{\lambda} + \bar{f} \odot \bar{\Delta}$ and $\bar{\nu} = \bar{\lambda} - \bar{\Delta}$ where $\odot$ is componentwise multiplication. Then the optimization problem becomes

$$\min_{\nu, \bar{\nu}} \sum_{i=1}^{n} v_i \quad \text{(12a)}$$

subject to

$$K^T \bar{\nu} = \bar{\Delta}^T \mu^* \quad \text{(12b)}$$

$$\nu \geq \bar{\nu} \odot \nu \quad \text{(12c)}$$

$$\nu \geq -\bar{\nu} \odot \nu. \quad \text{(12d)}$$

where the last two inequalities come from the nonnegativity constraint of $\bar{\Delta}$. Suppose $\nu, \bar{\nu}$ are optimal solutions. Because we are minimizing the sum of the components of $\nu$, $v_i = \bar{f}_i \max(v_i, -v_i)$. This is equivalent to $v_i = \bar{f}_i|v_i|$. \hfill $\Box$

### Appendix

In the example of power systems, a line in a power grid is characterized by its admittance (reciprocal of its reactance). The voltage at each of the nodes is described by a complex number with magnitude $V$ and angle $\theta$. The original OPF problem is nonlinear in its most accurate formulation, but with the mild DC optimal power flow approximation, the voltage magnitudes of all nodes are fixed and the angle differences drive the flow of power. In particular, if the admittance between nodes $i$ and $k$ is $b_{k,i}$, the power flow $f_{k,i}$ from $i$ to $k$ is $b_{k,i}(\theta_i - \theta_k)$. Consider the nodes $1, \ldots, m$ in a cycle of length $m$. Then

$$\sum_{j=1}^{m} f_{j,i+1} = \sum_{j=1}^{m} b_{j,i}(\theta_i - \theta_j) + \sum_{j=1}^{m} (\theta_j - \theta_{j+1}) = 0,$$

since all of the angles cancels out. Therefore, a weighted sum of the flows along a cycle must sum up to 0.

As described in the main text, due to the existence of these cycle constraints, the edge flows must lie in a subspace in $\mathbb{R}^m$. This space is the so-called fundamental flow space, with a dimension of $n - 1$ corresponding to $n - 1$ linearly independent equality constraints. The rest of the $m - n + 1$ flows are uniquely determined by the cycle constraints. There are many ways to construct this space. A simple construction is to select a spanning tree of the network, where the flows on the edges in this tree are fundamental and form the cycle basis of the network.

### A. Proof of Theorem 2

**Proof.** We give the analysis of the relationship between binding inequality constraints and optimal dual variables associated with equality constraints. With known $\mu^*$ and separable constraints (6b)(6c), the dual problem is decomposable to the optimization problem consisting $\bar{\lambda}, \bar{\Delta}$ and $\bar{\nu}, \nu$ respectively. Optimization involving $\bar{\nu}, \nu$ can be reformulated as

| Time(s) | NF 20-Node | NF 50-Node | NF 200-Node | OPF 14-Bus | OPF 39-Bus |
|---------|------------|------------|------------|------------|------------|
| Low     | 0.0008     | 0.0152     | 0.0009     | 0.0292     | 0.0108     | 0.1698     | 0.0013     | 0.0333     | 0.0022     | 0.0283     |
| Medium  | 0.0006     | 0.0147     | 0.0010     | 0.0267     | 0.0121     | 0.1622     | 0.0014     | 0.0233     | 0.0024     | 0.0285     |
| High    | 0.0006     | 0.0149     | 0.0010     | 0.0256     | 0.0108     | 0.1698     | 0.0015     | 0.0237     | 0.0028     | 0.0291     |

**TABLE IV**

**COMPUTATION TIME PER INSTANCE FOR NF PROBLEM AND OPF PROBLEM USING CVXOPT AND PROPOSED NEURAL DECODER.**
Our formulation showed in (1) is a general formulation of minimum-cost network flow problem. For instance, in the transportation problem, we could treat the nodes either sources or sinks of the product, and thus we can interpret the KKT conditions, it is also the optimal solution, which completes the proof.

Fig. 6. The example network for our simulations on network flow and OPF problem.

C. Proof of Lemma 2

We use the following lemma:

Lemma 3. For an LP problem \( \{x^* = \arg \min_x c^T x | Ax = b, x \geq 0 \} \) with \( A \in \mathbb{R}^{m \times n}, x \in \mathbb{R}^m, m < n \), let (1), (2), ..., (m) be the indices of selected columns of \( A \), such that \( Bx^* = b \) with \( B = [a_{(1)} \ldots a_{(m)}] \in \mathbb{R}^{(m \times m)} \) as an invertible basis. For any \( b = b + \delta \), the optimal solution is still given by \( B \) if and only if \( B^{-1}b + B^{-1}\delta \geq 0 \).

The proof of Lemma 2 follows from Lemma 3 by converting our LP of interest to the standard form. Despite the latter being a known result in linear programming, we have not been able to find the proof of the vector form in existing literature (the component by component form can be found in [1], [35]). Therefore we provide the following proof for completeness.

Proof. To find the region where \( \tilde{b} = b + \delta \) has the same set of active constraints at the optimal solution, denote the new optimal solution as \( \tilde{x}^* \) that satisfies \( A\tilde{x} = \tilde{b} \). So the new optimization problem involving \( \tilde{b} \) becomes

\[
\tilde{x} = \arg \min \ c^T x \\
\text{s.t.} \quad Ax = b + \delta \\
x \geq 0
\]

Since we have \( x^* = B^{-1}b \geq 0 \) with input \( b \), and since \( x^* \) and \( \tilde{x} \) can be represented by the same basis \( B \in \mathbb{R}^{m \times m} \), we have

\[
B^{-1}(b + \delta) \geq 0
\]

to ensure the optimal solution’s feasibility. So the resulting \( \delta \) must satisfy (14).

On the other hand, if \( B^{-1}(b + \delta) \geq 0 \), while \( \tilde{x} = B^{-1}(b + \delta) \) satisfies both equality and inequality constraints. By checking the KKT conditions, it is also the optimal solution, which completes the proof.

D. Modeling of Network Flow Problem

We note that formulation showed in (1) is a general formulation of minimum-cost network flow problem. For instance, in the transportation problem, we could treat the nodes either sources or sinks of the product, and thus we can interpret the nodal equality as \( \sum_{k=1}^n f_{jk} = a_j \) for all source nodes \( j(j = 1, 2, \ldots, m) \) and \( \sum_{j=1}^m f_{jk} = d_k \) for all destination nodes \( k(k = 1, 2, \ldots, n) \). In the shortest-path problem, \( C_i \) denotes the distance associated with each edge \( f_i \). With a net supply of one unit at the source, we have \( \sum_{k=1}^n f_{jk} = 1 \) for the source node \( j \), and \( \sum_{j=1}^m f_{jk} = 1 \) for the destination node \( k \), and the objective is to send one unit of flow from the source to the link at minimum cost.

E. Simulation Details and Ablation Study

In the data generation process, we collected the input data, the output optimal costs, the dual variables and optimal solutions for each network. We collected all the data samples which are feasible for optimization problem (1) and (2). With a nominal load level \( \ell \), in the standard network flow problem, we generate input samples from the uniform distribution \([0.9\ell, 1.1\ell], [0.8\ell, 1.2\ell], [0.7\ell, 1.3\ell] \); in the OPF problem, we generate input samples from the uniform distribution \([0.8\ell, 1.2\ell], [0.5\ell, 1.5\ell], [0.2\ell, 1.8\ell] \).

In Figure 6 we show the underlying graph for network flow and OPF simulation. When reporting the results on testing instances, We treat the following solutions as “infeasible”:
F. Computation Time Analysis

As is reported in Table IV, proposed Neural Decoder approach could achieve faster computation of optimal solutions for various linear programming problems. We note that we are making use of off-the-shelf packages for neural networks differentiation and equation solving. More acceleration may be expected if the specific neural network architecture or the equation structure is considered. In Fig. 7 we illustrate the computation time for each part of the Neural Decoder computing, and it can be seen the major computation burdens are coming from the linear equation solver.