# Robust Deep Learning for Wireless Network Optimization

Shuai Zhang*, Bo Yin*, Suyang Wang*, Yu Cheng*
Department of Electrical and Computer Engineering, Illinois Institute of Technology, Chicago, IL 60616

***Abstract*—Wireless optimization involves repeatedly solving difficult optimization problems, and data-driven deep learning techniques have great promise to alleviate this issue through its pattern matching capability: past optimal solutions can be used as the training data in a supervised learning paradigm so that the neural network can generate an approximate solution using a fraction of the computational cost, due to its high representing power and parallel implementation. However, making this approach practical in networking scenarios requires careful, domain-specific consideration, currently lacking in similar works. In this paper, we use deep learning in a wireless network scheduling and routing to predict if subsets of the network links are going to be used, so that the effective problem scale is reduced. A real-world concern is the varying data importance: training samples are not equally important due to class imbalance or different label quality. To compensate for this fact, we develop an adaptive sample weighting scheme which dynamically weights the batch samples in the training process. In addition, we design a novel loss function that uses additional network-layer feature information to improve the solution quality. We also discuss a post-processing step that gives a good threshold value to balance the trade-off between prediction quality and problem scale reduction. By numerical simulations, we demonstrate that these measures improve both the prediction quality and scale reduction when training from data of varied importance.**

***Index Terms*—multi-hop wireless mesh network, deep learning, network utility maximization**

## I. Introduction

Wireless optimization problems are challenging and ubiquitous in many applications. Network controllers have commonly followed a model-based approach: a mathematical model, parameterized by design variables, standard stipulations and control variables, is formed which gives a measure of the system utility or objective. Then given such a model, optimization problems are solved by a network entity, e.g., a base station or access point, and the optimal solutions are used as the control decision. Then this process repeats periodically to adapt to the time-varying conditions of the network. With the ever growing need for higher spectrum and power efficiency and lower delay, the next generation communication system will go through a fundamental change in its paradigm.

To tackle this challenge, studies on the wireless network optimization in the past years focus on the development of approximation algorithms [1]–[3]. Wireless networks are envisioned to be an integral part of many emerging applications, e.g., the Internet of Things (IoT), which are inherently dynamic and require adaptive control. In this way, the conventional approaches are becoming increasingly infeasible since problem instances are solved in a case-by-case manner. Inspired by the recent success of machine learning (ML) in other domains, data-driven approaches receive a lot of attention in the area of wireless network optimization.

One promising thread in wireless network optimization follows the paradigm of supervised learning [4]–[6]. Specifically, works in this thread aim to utilize the pattern matching capability of deep neural network (DNN) for distilling useful insight with respect to a specific optimization task from the historical problem instances that are solved by conventional algorithms. The trained neural network could *generalize* to new problem instances if the training data set is sufficiently large, and if the model has the representation power to characterize the relationship between the input and output data. It is possible to build an end-to-end learning framework in which the mapping from problem instances to solutions is approximated. However, for problems that involve highly sparse and high dimensional data, e.g., joint routing and scheduling decisions, directly learning the mapping is difficult in practice. Instead of building a ML model that outputs the solutions directly, the work in [7] circumvents this difficulty via training a DNN that learns meaningful properties of the solution space. More precisely, the approach proposed in [7] can identify the subspace in which an optimal or near-optimal solution is included with high probability. As the search space is narrowed down, the total amortized computation cost of the conventional approximation algorithm can be significantly reduced.

There are still several issues which hinder deep learning from gaining more adoption in the networking field. A commonly encountered problem is that not all training samples are of the same importance. This can be attributed to two sources. First is *class imbalance*: it means that the probability distribution to learn is severely distorted towards one type of outcome over another. For example, in a D2D link scheduling problem, most of the time links choose not to transmit. This can lead to severe bias in the final training result as the model is likely to output "not transmit" even when it is not supposed to, because on average this matches the training data and is easier to make this decision. In this sense, another sample of not transmitting is not of much use for the training progress. The other reason samples should not be treated equally is that the labels' quality can vary: the samples can be generated and collected under subtle, different circumstances; labels can be close to, but not exactly at the optimum of the problem to solve due to the implementation of the conventional algorithms.

978-1-7281-5089-5/20/$31.00 ©2020 IEEE

These minor factors in total can have an effect on the sample labels that are similar to random noise. It is entirely possible to train on these polluted data source and capture spurious patterns that are not present in the testing or application. Moreover, if the output needed by the network layer is discrete, but the neural network provides a continuous variable, then a proper threshold value should be established. In most works, there is no principled approach to selecting good threshold and the process is through simple trial-and-error, causing potential loss of performance.

Considering these issues, we propose a novel scheme to improve learning performance when the training samples are of varied importance. Our contribution can be summarized as follows:

- An adaptive sample weighting scheme. It learns the sample weights that can fit in any classification framework without new weighting function. The weight is learned based on how much each sample contributes to the metric function on a separate, high quality dataset.
- An improved loss measure based on the cross entropy function. Rather than treating each link as an equal and independent label, we consider *each link carrying a unit flow* as an independent label. In this way the link flow information is also used to guide the learning process.
- A threshold value selection criterion to further improve the performance. Based on Bayesian estimation, we use an exponential family probabilistic model which gets updated with samples of threshold values to avoid large overhead involved in evaluation.

## II. System Models and Problem Setting

We are interested in a multi-hop single-radio single-channel wireless network, which is the general case of a D2D network. A set of communication nodes $\mathcal{N}$ is randomly distributed within a rectangular area. The time-slotted system has one given frequency band $W$, assuming synchronization between the nodes. A central node has access to the nodes' location information, and makes scheduling and routing decisions. At a given time slot, a non-determined number of pairs of nodes need to transmit to each other. The set of communication links is denoted by $\mathcal{E}$. The transmission capacity of the link between transmitter $u$ and receiver $v$ is denoted by $c(u, v)$. The multi-commodity flow demands are denoted by $\mathcal{K}$, in which each flow $k \in \mathcal{K}$ is a source and destination tuple: $k = \{s_k, t_k\}$. The system model is illustrated in Fig. 1.

In order to maximize the throughput-based system performance, at each slot the decisions need to make are:

- *scheduling*, determining which subset of links to activate to avoid interference;
- *flow allocation*, specifying which type of flow and how much of it should be transported on each activated link such that the flow conservation and link capacity is satisfied.

We consider the communication links to follow the protocol interference model [8]. This means a transmission from node $u$ to node $v$ is successful only if no other nodes within the detection range of $v$ is simultaneously transmitting. Moreover, each node works in a half-duplex manner, which prohibits the concurrent activation of links that involve overlapping nodes. The conflict relations among all the links can be characterized by a conflict graph.

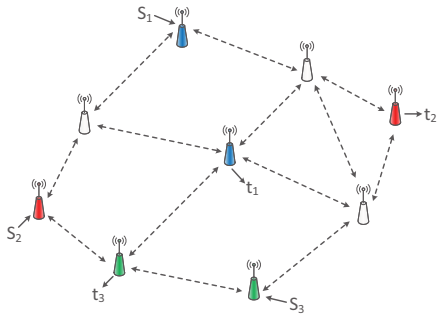



Fig. 1: System illustration

We adopt the pattern-based formulation in [9], where a transmission pattern is an independent set of the conflict graph. Formally, each pattern is characterized by a $|\mathcal{E}|$-dimensional vector, where the $j$th component equals to the feasible capacity of the $j$th link in this pattern. More precisely, if link $(u, v)$ is active in pattern $m$, the value of its corresponding component, denoted by $p_m(u, v)$, is $c(u, v)$; otherwise, it equals to 0. To ensure interference-free communications, all the transmission patterns are supposed to be scheduled in a TDMA manner. Let $\mathcal{M}$ be the set of all the transmission pattern and $\alpha_m$ denotes the fraction of slot allocated to pattern $m$.

Let $f_k(u, v)$ denote the flow allocation variable with respect to commodity $l$ over link $(u, v)$. Generally, all the flow variables need to satisfy following constraints.

- Link capacity: the sum of all the flows over a link does not exceed its capacity, i.e.,

$$\sum_{k \in \mathcal{K}} f_k(u, v) \leq c(u, v), \quad \forall (u, v) \in \mathcal{E} \tag{1}$$

- Flow conservation: for commodity $k$, the amount of flow entering an intermediate node equals to that exits the node, i.e.,

$$\sum_{u \in \mathcal{N}} f_k(u, v) = \sum_{u \in \mathcal{N}} f_k(v, u), \quad \forall v \neq s_k, t_k; \forall k \in \mathcal{K} \tag{2}$$

The interference-free requirement introduces following constraints.

$$\sum_{k \in \mathcal{K}} f_k(u, v) \leq \sum_{m \in \mathcal{M}} \alpha_m p_m(u, v), \quad \forall (u, v) \in \mathcal{E} \tag{3}$$

$$\sum_{m \in \mathcal{M}} \alpha_m = 1 \tag{4}$$

With the demand of commodity $k$ being expressed as

$$d_k = \sum_{v \in \mathcal{N}} f_k(s_k, v) - \sum_{v \in \mathcal{N}} f_k(v, s_k), \tag{5}$$

the maximum multi-commodity flow problem in wireless networks can be formulated as

$$\begin{aligned} \underset{\{f_k(u,v)\},\{a_m\}}{\text{Maximize}} \quad & \sum_{k \in \mathcal{K}} d_k \\ \text{s.t.} \quad & \text{constraints } (1), (2), (3), (4) \\ & f_k(u,v) \geq 0, \quad \forall (u,v) \in \mathcal{E}, k \in \mathcal{K} \\ & \alpha_m \geq 0, \quad \forall m \in \mathcal{M} \end{aligned} \tag{6}$$

The problem described above has the form of linear programming, because the objective and constraints are linear functions. However, it is easy to see that to obtain a set of non-interfering tuple links is equivalent to finding a graph coloring of links. Therefore the problem is essentially a mixed-integer linear programming (MINLP) type with an exponential number of variables, since each transmission pattern requires a time variable in the formulation Eq. (4). It is not practical to enumerate all patterns and use a standard linear programming solver. Practical algorithms typically employ the column generation method [10], which starts from an initial set of transmission patterns, solve a partial problem and use the dual solution to generate new pattern to add in the problem. Following this procedural column generation, a solution sufficiently close to the optimum solution to the original problem is obtained. In this way the algorithm memory usage is saved and complexity can be controlled as a trade-off with the objective. The reader is referred to the work [9] for a more detailed account of this method, which we use as a teacher algorithm in the later sections. The key intuition behind it is that the final optimum result only makes use of a very small subset out of all the transmission patterns, and most of the other patterns are given zero time share.

## III. Machine-learning based Scheduling

In this section we explain how supervised learning paradigm is used to alleviate the problem's complexity and explore the aspects where the deep learning could be more efficient.

### A. Supervised Learning for Problem Scale Reduction

Consider the combinatorial nature of problem (6), we adopt the framework proposed in [7] to develop our ML model, in which intermediate characterizations of the problem instances, i.e., link usefulness, are learned to reduce the problem scale. Roughly speaking, a link is considered to be useful if it is activated to accommodate arbitrary commodity flows. It is shown in [7] that the computational complexity of the column generation based optimization can be amortized significantly through removing those links that are probably useless.

In this framework, the ML model is trained in a supervised learning fashion. Formally, the training dataset $D_{\text{train}} \triangleq \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_i$ captures the optimization results of past problem instances. $\mathbf{x}^{(i)}$ denotes the vector representation[1] of problem instance $i$ and $\mathbf{y}^{(i)}$ characterizes the ground truth in terms of the link usefulness in the solution to this instance, i.e.,

$$\mathbf{y}^{(i)} = \{\mathbb{1}_{(\sum_{k \in \mathcal{K}^{(i)}} f_k^{(i)}(u,v) > 0)} : (u,v) \in \mathcal{E}^{(i)}\}, \tag{7}$$

where $\mathbb{1}_{(\cdot)}$ is the indicator function and superscript $(i)$ is used to index the problem instance. A DNN is trained over $D_{\text{train}}$ to learn a reasonable mapping from $\mathbf{x}^{(i)}$ to $\mathbf{y}^{(i)}$. Given a new problem instance with representation vector $\mathbf{x}^{(j)}$, the output of the DNN, say $\hat{\mathbf{y}}^{(j)}$, indicates the activation probabilities of the communication links. With the knowledge of this predicted result, the network topology of instance $j$ can be pruned properly to generate a reduced-size problem instance, which can be solved more efficiently by conventional algorithms.

During the training process, the parameters of the DNN are updated iteratively via the backpropagation method, with the objective of minimizing a loss measure $\mathcal{L}$ that quantifies the discrepancy between the predicted output and the ground truth. Typically, the loss measure $\mathcal{L}$ needs to be continuous and differentiable, since the the backpropagation method is gradient-based. However, task related performance measures may not always have such a property. For example, if the target $\mathbf{y}^{(i)}$ is a discrete variable representing categories, then in such classification problems, commonly used measures such as accuracy, precision and F scores are not differentiable. Note that $\mathbf{y}^{(i)}$ in $D_{\text{train}}$ is a binary vector, the link prediction can be considered as a multi-label binary classification task. In this way, a natural loss function is the average cross-entropy loss, i.e.,

$$\begin{aligned} \mathcal{L}(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)}) = -\frac{1}{|\mathcal{E}^{(i)}|} \sum_{n=1}^{|\mathcal{E}^{(i)}|} & (\hat{y}_n^{(i)} \log(y_n^{(i)}) \\ & + (1 - \hat{y}_n^{(i)}) \log(1 - y_n^{(i)})), \end{aligned} \tag{8}$$

where $\hat{y}_n^{(i)}$ and $y_n^{(i)}$ respectively denote the $n$-th component of $\hat{\mathbf{y}}^{(i)}$ and $\mathbf{y}^{(i)}$.

### B. Adaptive Sample Weights

Recall that in supervised learning, given the training dataset $D_{\text{train}} \triangleq \{(\mathbf{x}_i, \mathbf{y}_i)\}_i$, the training process can be expressed by the optimization problem

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \ \frac{1}{|D_{\text{train}}|} \sum_{i=1}^{|D_{\text{train}}|} \mathcal{L}(\hat{\mathbf{y}}^{(i)}(\boldsymbol{\theta}), \mathbf{y}^{(i)}) \tag{9}$$

$$\hat{\mathbf{y}}^{(i)}(\boldsymbol{\theta}) \triangleq \phi(\mathbf{x}^{(i)}; \boldsymbol{\theta}) \tag{10}$$

where $\hat{\mathbf{y}}^{(i)}$ is the predicted result given the input $\mathbf{x}^{(i)}$ produced by the neural network $\phi$ parameterized by $\boldsymbol{theta}$. After training, the parameterized neural function should minimize the expected loss measure $\mathcal{L}$ across all sample points.

This scheme works well when all the training samples have consistent importance, so all the samples contribute equally

[1] $\mathbf{x}^{(i)}$ can be either meta-features of the problem instance or intrinsic structures that are extracted via representation learning approaches.

when calculating the gradients of the output with respect to the model parameters. However, if a large portion $N$ of the samples, denoted as $D \subset D_{train}$ are labeled with corruptions, and only a small subset $D_{hq} \subset D_{train}$ of $M$ ($M \ll N$) samples can be verified to have high-quality labels, it is no longer appropriate to give them equal weight in the parameter updating. But it would also be a waste of data if one only trains with $D_{hq}$, and the model quality could suffer due to a smaller dataset. The better approach would be to give samples individual weights according to its importance. It is possible to identify and manually lower the weight of the low quality sample points before the training; but this requires the user to supply a weighting scheme independent of the learning progress, and the selection of a good scheme is essentially optimizing new hyperparameters requiring careful tuning before it can be integrated into an existing workflow.

Different from that, we propose to use a learning approach where a sample weighting scheme is generated from the high-quality data *during* the training process. Adapted from a mete-learning perspective [11], this approach fits into existing supervised learning framework widely used in networking applications, without requiring significant changes to the steps in a typical learning process.

We hope to derive the sample-wise weight coefficients $\mathbf{w} = \{w_i\}_i$, where the index $i$ is used in the dataset $D$ and each weight is a non-negative number. It is required to satisfy the following relationship:

$$\boldsymbol{\theta\prime}(\mathbf{w}) = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} w_i \mathcal{L}(\hat{\mathbf{y}}^{(i)}(\boldsymbol{\theta}), \mathbf{y}^{(i)}) \tag{11}$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\geq\mathbf{0}} \frac{1}{M} \sum_{i=1}^{M} \mathcal{J}(\hat{\mathbf{y}}^{(i)}(\boldsymbol{\theta\prime}(\mathbf{w})), \mathbf{y}^{(i)}), \tag{12}$$

where $\mathcal{J}$ is a continuous function that can be an identical to or different from the loss $\mathcal{L}$.

The above relationship comes from the following intuitive observation: for each sample weight vector $\mathbf{w}$, when it is applied in a training step on the set $D$, there exists an updated set of model parameters $\boldsymbol{\theta\prime}(\mathbf{w})$, obtained from the optimizer process, as $\mathbf{w}$'s function. We can test this new model parameter's performance by a *weight penalty* function $\mathcal{J}$ on the high-quality dataset $D_{hq}$, and since the labels can be trusted to be good, the $\mathcal{J}$ function's value should be a good indicator of the new parameter's quality unaffected by any degradation from the data. Notice that the $\mathcal{J}$ function can be non-differentiable, if its form permits finding a best $\mathbf{w}$ efficiently. And because $\mathcal{J}$ value is a function of the sample weights, one can find the corresponding best $\mathbf{w}$ which minimizes it.

However, although the mentioned best sample weights exist, practically it is not always feasible to attempt to find them directly. This is because the set of training samples can be very large, and to solve the equations to accuracy would need several rounds through it. Besides, today's learning process performs parameter updates with mini-batches of sample data rather than one sample at a time, so the above relationship should be adapted to accommodate it. Another relevant factor is that the new parameter is likely to not be a linear function of the gradient in many optimizer algorithms used today, e.g., Adam-like optimizers that make use of higher-order gradient statistics to regulate the effective learning rate, making it even more difficult to find the best sample weight.

To make the process practical in the presence of the above difficulties, we include the two additional approximating assumptions:

- the new parameter $\boldsymbol{\theta\prime}$ can be approximated by a linear function of the sample weights $\mathbf{w}$, and the penalty function $\mathcal{J}$ is differentiable w.r.t to $\mathbf{w}$;
- instead of finding the best $\mathbf{w\prime}$, in each iteration use a fixed number of gradient descent step to find a good enough $\tilde{\mathbf{w}}\prime$.

The above assumptions enables us to obtain differentiated weighting without incurring a large number of iterative updates to $\mathbf{w}$, as we observed in the numerical experiments. With the first assumption, one can directly use gradient descent to improve current $\mathbf{w}$; the second assumption uses the certain number gradient steps to get a surrogate of the optimum $\mathbf{w}$.

We can observe the relationship between the gradient with respect to each sample weight to understand the behavior:

$$\boldsymbol{\theta\prime}(\mathbf{w}) = \boldsymbol{\theta} - \eta_{\text{eff}} \nabla_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} w_i \mathcal{L}(\phi(\mathbf{x}^{(i)}, \boldsymbol{\theta}), \mathbf{y}^{(i)}) \tag{13}$$

$$\nabla_{w_i} \boldsymbol{\theta\prime} = -\eta_{\text{eff}} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\phi(\mathbf{x}^{(i)}, \boldsymbol{\theta}), \mathbf{y}^{(i)}) \tag{14}$$

$$\begin{aligned} \nabla_{w_i} \sum_{i=1}^{M} \mathcal{J} &= \frac{1}{M} \sum_{i=1}^{M} \nabla_1 \mathcal{J} \cdot \nabla_{\boldsymbol{\theta}} \phi \cdot \nabla_{w_i} \boldsymbol{\theta\prime} \\ &= -\frac{1}{M} \sum_{i=1}^{M} \nabla_{\boldsymbol{\theta}} \mathcal{J}(\phi(\mathbf{x}^{(i)}, \boldsymbol{\theta}), \mathbf{y}^{(i)}) \cdot \eta_{\text{eff}} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\phi(\mathbf{x}^{(i)}, \boldsymbol{\theta}), \mathbf{y}^{(i)}) \end{aligned} \tag{15}$$

From Eq. (15), it is clear that when the two gradient expression is similar, i.e., having a large product, then the weight will become larger, since the weight is updated with the negative of gradient. At the same time, the update is still regulated with the effective step length $\eta_{\text{eff}}$, so adjustable learning rate plays a role here, adjusting how much the sample weight is changed each time these steps are executed. In the context of network link evaluation, this would mean that the each sample in the set $D$ is weighted according to how similar the gradient flow it causes to the loss $\mathcal{L}$ compared with the average gradient caused by the high-quality samples to the penalty function $\mathcal{J}$.

### C. Choice of Penalty Function $\mathcal{J}$ in link prediction

From previous analysis we have seen that the weights are based on gradient similarity, then it is obvious that use $\mathcal{J} = \mathcal{L}$ is acceptable. However, if in the high quality set $D_{\text{hq}}$, there are additional network layer information available, we can choose a different $\mathcal{J}$ function to take advantage of that.

If cross entropy is chosen, then for any graph instance $g$, minimizing the cross-entropy from the predicted link usage

to the actual link usage is equivalent to maximizing the probability

$$\prod_{i \in \mathcal{E}_g} \hat{y}_i^{y_i} (1 - \hat{y}_i)^{1 - y_i}. \tag{16}$$

Such a probability measure contains an implicit assumption that whether the prediction is right or wrong contributes equally to how the model parameters are adjusted. This assumption is valid when the underlying objects in the predictions are largely unrelated and equally important; the outputs can be said to be "structure-less". But if the outputs here correspond to the links in a network, it is no longer appropriate because the links due to the graph topology have different topological importance. For example, some links can act as bottlenecks that are part of a minimum cut, so wrong predictions at this links are likely to cause the prediction quality to suffer. In this case, the output labels contain a structure that should reflect in the design choice of the penalty function.

Considering this, suppose that in $D_{hq}$, in addition to label, there exists another information vector $\mathbf{z} \in \mathbf{R}^+$ that is correlated to $\mathbf{y}$ and holds information about the structural importance of each output label. The new penalty function can take the form

$$\prod_{i \in \mathcal{E}_g} \hat{y}_i^{(1+\beta \text{sigmoid}(z_i)) y_i} (1 - \hat{y}_i)^{(1+\beta \text{sigmoid}(z_i))(1 - y_i)}. \tag{17}$$

Now each link with importance $z_i$ is treated as $(1 + \beta \text{sigmoid}(z_i)$ copies stacked together. This scheme favors the link more important structurally.

### D. Threshold value optimization

The neural model as we have shown is still a continuous one. In the training phase, we could use continuous proxy measures such as cross-entropy that measures how good the decision is. But when the model is put to use, the only output that matters is the discrete decision variables. Converting from a continuous output to a discrete variables is typically done through the use of threshold values: if the final output $\hat{y}$ is one of the 2 possibilities, then $\hat{y} = 0$ if $\hat{y} \geq p$ and 1 otherwise.

In our case, the threshold value has a large impact on the final system performance. There are two key performance metrics used:

- approximation ratio $r$, expressed as the ratio of reduced problem instance optimization objective value to the value of the original problem.
- time cost reduction $d_t$. This measures the fraction of run time saved by solving a smaller problem instance. Note that it also considers the additional run time caused by the loss of connectivity: if the reduced problem instance does not have a feasible solution, then the algorithm will attempt to add back links that are not present, in the order of link score.

For easy comparison, we use their sum as the scalar value called *solution merit*: $\xi(\alpha) = \mathbb{E}\, r(\alpha) + d_t(\alpha)$, where $\alpha$ is the threshold value to be chosen and the expectation is taken over problem instances. Then finding a good threshold value is equivalent to maximizing the merit value. The relationship between the threshold value and the merit value is not a straightforward one. A lower threshold value includes more links in the solution, and it increases the approximation ratio and at the same time reduces the time cost reduction because the problem has cut off less links. Similarly, a higher threshold value prunes off links more aggressively, and this causes solution quality to be likely lower, and may decrease the solution time, because the pruning may cause the system to become infeasible and additional time is needed to revert the instance to a feasible one.

Typically to choose a good threshold value to is through trial-and-error: one can use a grid search, where the neighborhood of valid threshold values is divided into small regions and a best region can be picked by evaluating them all. However, considering the inherent structure of this problem, we adopt a Bayesian approach for finding good threshold values. This method is a fitting one because the relationship between the merit and the threshold value is not explicitly expressible with closed-form expressions and thus hard to apply typical optimization techniques; Moreover, it is expensive to evaluate even one point of this relationship, since we have to iterate over all the points in $D_{hq}$ for every $\alpha$. The Bayesian approach maintains an internal probabilistic model whose *mean* approaches $\xi(\alpha)$ as more samples are observed, thus making it efficient to rapidly finding a good $\alpha$ given limited knowledge of the function.

In this iterative algorithm, first we assume that without further information, the $\alpha$ is described with a type of distribution $P_0(\alpha; \gamma)$ within a specified range. We start with a set of observation points in the form of $(\alpha_i, \xi(\alpha_i))_i$, then fit the distribution according to these observations. With the updated probabilistic model $P\prime$ of $\alpha$, the next point to sample $\alpha\prime$ is given by the *expected improvement* [12] method:

$$\alpha\prime = \arg\max_{\alpha} \mathop{\mathbb{E}}_{P\prime} max(\xi(\alpha) - \xi(\alpha^+), 0), \tag{18}$$

where $\xi(\alpha^+)$ is the largest value observed so far. This step uses the probabilistic model to estimate the expected improvement over the current best value, and returns the currently unobserved point $\alpha\prime$ to evaluate next. The algorithmic steps are summarized in the following listing.

## IV. Numerical Results

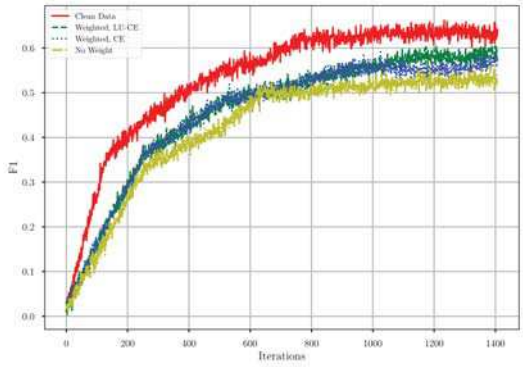
We implement this system with the existing software framework Pytorch [13] in Python, and conducted experiments to test its system performance. We consider the testing network to be located within a square area of $1000$ meters sides, with its illustration shown in Fig. 1 and the parameters listed in Table I. The communication nodes are randomly distributed with a minimum separation of $0.5$m. Two nodes are connected by a link if their distance is smaller than the system parameter *transmit range* and links present as an interference if their distance is smaller than *detection range*. Once the

**Algorithm 1:** Threshold Generation

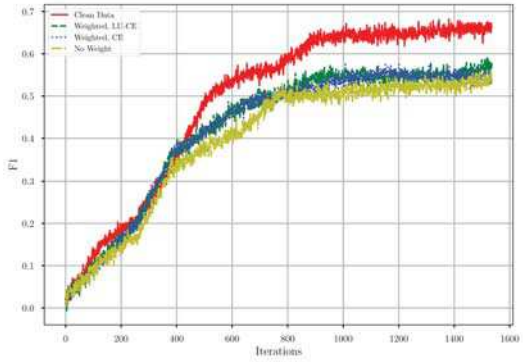
**input :** $P_0(\alpha)$, range of $\alpha$, $\xi$, MaxIter, $n_0$
**output:** threshold value $\alpha^*$
Sample and Evaluate $n_0$ points in the range of $\alpha$ randomly
Let $i$ = 1
**while** $i \leq$ *MaxIter* **do**
  Let $P_i$ be the updated distribution of $\alpha$ with all observed data
  $\alpha_i = \arg\max \zeta(\alpha; P_i)$
  Record $(\alpha_i, \xi(\alpha_i)$
**end**

| | |
|---|---|
| experiment area | square 1000m, 1000m |
| transmit range | 100m |
| detection range | 200m |
| Number of Nodes | [16, 64] |
| Train, Validation, Test | 10000, 1500, 500 |
| Batch size | 16 |
| $\beta$ | 3 |
| Probabilistic Model | Gaussian Process Matern kernel |
| Kernel Parameter | Scale=1, nu=3.5 |

TABLE I: List of Parameters Used




(a) 16 nodes




(b) 64 nodes

Fig. 2: The F1-score vs Training iterations

node positions are calculated, the link capacity is taken as the Shannon capacity with no interference and only white noise.

We generate data at two network scales, 16 nodes and 64 nodes respectively. For each scale we use 5 different node location configurations and average the performance results. For each setting, we use conventional teacher algorithm listed in reference [14] to generate the link usage information for 12000 instances. These instances' inputs are different by the traffic demand and random perturbations to the link capacities. Among the training samples, we add random noise to the link information such that around 5% of the links receive wrong labels. The validation and test sets contain samples where no errors are introduced. The difference between the validation In implementing the adaptive weight generation, we only use one gradient step in obtaining $\mathbf{x}w$ and adopt a small batch size, which is not optimized in order to save time on the training progress. For the threshold optimization, the Matern kernel [15] is used, which is a generalized Gaussian radial basis function commonly used in probabilistic models.

In Fig. 2 we show the training progress plot, demonstrating the effectiveness of adding adaptive sample weights to increase the final validation performance. We can see that across the two network scales, in the presence of varied quality data, the general trend is that the performance metrics has a marked decrease without additional measures, and the training progress is also slower. With adaptive weights, such degradation can be alleviated partially, although there is still a gap from the performance gained from clean data.

Regarding the choice of penalty function, we find that by adding additional network-layer information, there is a moderate increase in the prediction accuracy. This is expected because in our problem, the output labels have a hidden structure and therefore loss function should reflect that fact.

In Fig. 3 we show how the optimum threshold is estimated by the Bayesian method. Several estimated $\xi$ function, obtained as the mean of the Matern process, is plotted in the same frame. We can see that with more sampled $\alpha$, $\xi(\alpha)$ pairs, the estimated function converges to a reasonable estimation. Another way of choosing a good threshold value is to choose $\alpha$ such that the F1 score is maximized [16]. We also compare the rule-of-thumb of choosing 0.5 as the threshold because the output is a real number between 0 and 1. The results clearly indicate that our approach is superior for finding a better threshold. For small variations of $\alpha$, the solution merit value can still demonstrate significant difference in the final test performance.

## V. Summary

We propose a data-driven deep neural networks-based approach in the wireless optimization task to identify important network links in scheduling and multi-hop flow routing. To make the learning process robust in the presence of varied-quality data, we develop an adaptive sample weight scheme, and use a improved penalty function and threshold selection method to enhance the learning results from such data. The

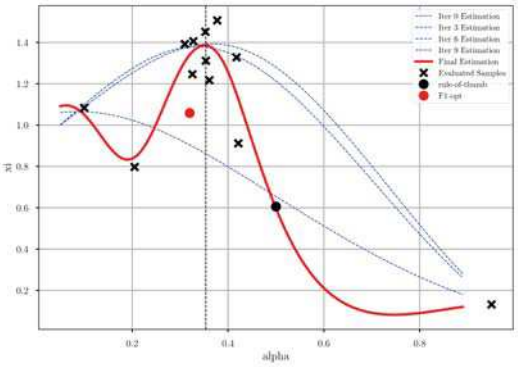



Fig. 3: The convergence of the Bayesian estimation to the best $\alpha$ value, marked by the vertical line. All the $\alpha$ values evaluated are marked with "x". The other methods' solutions are marked with red and black dot.

simulation results confirm that these measures combined result in a higher solution quality at the cost of additional processing. As data becomes a more pressing issue, future work is expected to explore techniques for the learning model to deal with the issues related to it.

## REFERENCES


[1] D. Chafekar, V. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan, "Approximation algorithms for computing capacity of wireless networks with sinr constraints," in *Proc. of IEEE INFOCOM*, 2008, pp. 1166–1174.

[2] S. Misra, S. D. Hong, G. Xue, and J. Tang, "Constrained relay node placement in wireless sensor networks: Formulation and approximations," *IEEE/ACM Transactions on Networking*, vol. 18, no. 2, pp. 434–447, 2010.

[3] R. Gandhi, Y.-A. Kim, S. Lee, J. Ryu, and P.-J. Wan, "Approximation algorithms for data broadcast in wireless networks," *IEEE Transactions on Mobile Computing*, vol. 11, no. 7, pp. 1237–1248, 2012.

[4] M. A. Wijaya, K. Fukawa, and H. Suzuki, "Neural network based transmit power control and interference cancellation for mimo small cell networks," *IEICE Transactions on Communications*, vol. 99, no. 5, pp. 1157–1169, 2016.

[5] F. Tang, B. Mao, Z. M. Fadlullah, N. Kato, O. Akashi, T. Inoue, and K. Mizutani, "On removing routing protocol from future wireless networks: A real-time deep learning approach for intelligent traffic control," *IEEE Wireless Communications*, vol. 25, no. 1, pp. 154–160, 2017.

[6] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for interference management," *IEEE Transactions on Signal Processing*, vol. 66, no. 20, pp. 5438–5453, 2018.

[7] L. Liu, B. Yin, S. Zhang, X. Cao, and Y. Cheng, "Deep learning meets wireless network optimization: Identify critical links," *IEEE Transactions on Network Science and Engineering*, 2018.

[8] P. Gupta and P. R. Kumar, "The capacity of wireless networks," *IEEE Transactions on information theory*, vol. 46, no. 2, pp. 388–404, 2000.

[9] Y. Cheng, X. Cao, X. S. Shen, D. M. Shila, and H. Li, "A systematic study of the delayed column generation method for optimizing wireless networks," in *Proceedings of ACM MobiHoc*, 2014, pp. 23–32.

[10] D. Bertsimas and J. N. Tsitsiklis, *Introduction to linear optimization*. Athena Scientific Belmont, MA, 1997, vol. 6.

[11] M. Ren, W. Zeng, B. Yang, and R. Urtasun, "Learning to reweight examples for robust deep learning," *arXiv preprint arXiv:1803.09050*, 2018.

[12] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[13] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[14] L. Liu, X. Cao, Y. Cheng, and Z. Niu, "Energy-efficient sleep scheduling for delay-constrained applications over WLANs," *IEEE Transactions on Vehicular Technology*, vol. 63, no. 5, pp. 2048–2058, Jun. 2014.

[15] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*, Springer, 2003, pp. 63–71.

[16] Z. C. Lipton, C. Elkan, and B. Naryanaswamy, "Optimal thresholding of classifiers to maximize f1 measure," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2014, pp. 225–239.