LEARNING TO REFIT FOR CONVEX LEARNING PROBLEMS

Yingyan Zeng  
Virginia Tech  
yingyanzeng@vt.edu  

Tianhao Wang  
Princeton University  
tianhaowang@princeton.edu  

Si Chen  
Virginia Tech  
chenxi@vt.edu  

Hoang Anh Just  
Virginia Tech  
just@vt.edu  

Ran Jin  
Virginia Tech  
jran5@vt.edu  

Ruoxi Jia  
Virginia Tech  
ruoxijia@vt.edu

ABSTRACT

Machine learning (ML) models need to be frequently retrained on changing datasets in a wide variety of application scenarios, including data valuation and uncertainty quantification. To efficiently retrain the model, linear approximation methods such as influence function have been proposed to estimate the impact of data changes on model parameters. However, these methods become inaccurate for large dataset changes. In this work, we focus on convex learning problems and propose a general framework to learn to estimate optimized model parameters for different training sets using neural networks. We propose to enforce the predicted model parameters to obey optimality conditions and maintain utility through regularization techniques, which significantly improve generalization. Moreover, we rigorously characterize the expressive power of neural networks to approximate the optimizer of convex problems. Empirical results demonstrate the advantage of the proposed method in accurate and efficient model parameter estimation compared to the state-of-the-art.

1 Introduction

In many essential applications, machine learning (ML) models need to be trained repeatedly on different subsets of the same dataset. For example, Shapley Value (SV) [Jia et al., 2019a,b,c, Wang et al., 2020, Ghorbani and Zou, 2019] has been widely employed to fairly value the contribution of each training data point. The number of models to be retrained on subsets grows exponentially in the number of points for exact SV calculation, which motivates the extensive study on estimating SV efficiently. However, considering the important application of SV such as guiding the allocation of monetary rewards to data providers in a collaborative learning process, a small estimation error of SV may cause huge unfairness, mapping to monetary value with a large scale. Another example is tracking the responsibility of data providers for model performance by SV, which is also very sensitive to the SV estimation error [Ghorbani and Zou, 2019]. Given the target upper bound of estimation error $\epsilon$ in $L_2$-norm, existing estimation methods require training on $O(N \log N/\epsilon^2)$ subsets, which easily becomes impractical when $\epsilon$ is small [Jia et al., 2019a]. Therefore, the efficient model retraining on a large number of subsets is essential for accurate SV estimation.

Besides SV, retraining models on subsets of data is also used to diagnose the variation of learning results [Giordano et al., 2019a,b] and estimate predictive uncertainty [Lakshminarayanan et al., 2016]. In summary, these applications pose a computational challenge when the number of training data points is large and when the computation complexity of training a model from scratch is high.

To achieve efficient model updating, post-hoc approaches have been proposed to estimate the change of model parameters in response to a dataset change. For instance, influence functions [Koh and Liang, 2017] approximate the model change using a first-order Taylor’s approximation of the loss function, and hence can be implemented efficiently to understand the effect of small changes on training data, e.g., removing a single point. However, these approaches become inaccurate when estimating the impact of a large change on the training data. Different from the post-hoc approximation methods, learning to optimize (L2O) attempts to learn or even generate a new optimization method for an optimization problem in a data-driven manner. Despite promising empirical results, which show learned optimization
algorithms are often much faster than manually designed algorithms [Andrychowicz et al., 2016, Li and Malik, 2016, Chen et al., 2017, Lv et al., 2017, Wichrowska et al., 2017, Metz et al., 2019, Cao et al., 2019, Chen et al., 2020a, b]. Therefore, we propose to learn to estimate the model parameters by a deep neural network (DNN) for repeated training of subsets with large changes. In this work, we focus on convex learning problems, and develop a model-agnostic method OPTLEARN to directly estimate the optimized model parameters for any given input data set. We introduce two regularization terms, namely, KKT regularizer and utility regularizer, to impose the learned models to satisfy optimality conditions and maintain utility, respectively.

We further provide the theoretical analysis to support the effectiveness of the proposed method. Specifically, we explore the sufficient conditions such that the mapping between the dataset and the corresponding learned model parameters could be efficiently approximated by DNNs. We conduct extensive experiments, and the results demonstrate the efficiency and effectiveness of the proposed method compared to the state-of-the-art.

2 Related Work

2.1 Rapid Model Parameter Approximation. Machine unlearning and incremental model maintenance provide post-hoc techniques to estimate model parameters without retraining from scratch. Ginart et al. [2019], Nguyen et al. [2020], Brophy and Lowd [2021] have investigated unlearning strategies on specific types of learning algorithms, whereas Bourboule et al. [2021] provides generally applicable strategies. For simple linear models, Cauwenberghs and Poggio [2001], Schelter [2019], Wu et al. [2020a] introduce incremental model maintenance techniques for efficient updating a model for both data deletion and addition. In deep neural networks, influence function [Koh and Liang 2017] is proposed to measure the effect of a manipulated data point by using Taylor expansion to approximate model parameters. DeltaGrad [Wu et al., 2020b] saves optimizer’s update steps in order to more accurately approximate the removal of multiple sample points. Golatkar et al. [2020] used the approximation of the Fisher Information Matrix for the remaining data sample to measure the update step to modify the model parameters. Although all techniques are capable of efficient model parameter approximation for a change with small number of samples [Basu et al. 2020, Mahadevan and Mathioudakis, 2021], they are not scalable for situations when a large number of training points are added, deleted, or altered.

Learning to Optimize. L2O focuses on learning the optimization algorithm (i.e., optimizer). An early approach was provided in [Andrychowicz et al., 2016] to use Recurrent Neural Networks (RNN), especially LSTM models, to learn the optimizer. For larger model training, it requires the LSTM model to iterate through more time steps. Unfortunately, that will create a vanishing gradient, an exploding gradient, or memory insufficiency of the LSTM optimization, which in turn will render unstable training of L2O. Due to these barriers, more works [Chen et al., 2017, Lv et al., 2017, Wichrowska et al., 2017, Metz et al., 2019, Cao et al., 2019, Chen et al., 2020a, b] focus on overcoming those problems by improving the LSTM structure. Even though, improved LSTMs are able to skip the “bottleneck”, those works focus mainly on specific optimizers’ family, such as SGD, Adam, or RMSProp. Another branch of L2O technique was developed in parallel. Li and Malik [2016] proposed reinforcement learning (RL) technique, in which the RL policy is the update step of the optimizer and the reward is the loss for the optimizer. However, RL methods share similar problems as LSTM models; they are not scalable. Therefore, Almeida et al. [2021] proposed to update the optimizer’s hyperparameters instead of learning to update parameters, which helps to improve the generalization ability of L2O models. Despite the advantages in fast optimization and potential generalizability, L2O learns to learn and optimize the optimizer, which is still computationally intensive to be applied to repeated model training on large number of subsets.

3 Approach

3.1 Preliminaries

Given a data point $(x, y)$, where $x \in \mathbb{R}^d$ is the feature vector and $y \in \mathbb{R}$ is the label, a model $f(x; \theta)$ parameterized by $\theta \in \mathbb{R}^{\text{param}}$ maps the input feature $x$ to the model’s prediction $\hat{y}$. A loss function $\ell(\theta; (x, y)) = \ell(f(x; \theta), y)$ measures the discrepancy between the prediction $\hat{y} = f(x; \theta)$ and the true value $y$. Denote $D = \{(x_i, y_i)\}_{i=1}^n$ as the full training data set containing all $n$ training data points and $D_S$ as a subset with $n_S$ data points sampled from $D$. For an empirical risk minimization (ERM) problem, the parameters are optimized to minimize the training loss of the subset $D_S$ as:

$$
\min_{\theta_S \in \mathbb{R}^{\text{param}}} L(\theta_S; D_S) = \frac{1}{n_S} \sum_{i=1}^{n_S} \ell(\theta_S; (x_i, y_i)) + \frac{\lambda}{2} \| \theta_S \|_2^2
$$

(1)
where $\lambda$ is a hyperparameter that controls the degree of penalization on the $L_2$-norm of the estimated parameters to increase the model’s generalizability. It is a common case in ML that the training loss $L$ is convex in $\theta$, and thus $[1]$ becomes a convex minimization problem. For common convex problems, a solver of the learning algorithm is designed to optimize the problem $[1]$ and obtain the optimized parameter $\hat{\theta}_S$ for the training set $D_S$. In general, we define parameter function $A$ as the mapping from the input training set $D_S$ to the optimized parameter $\hat{\theta}_S$ in terms of certain learning algorithms, i.e., $\hat{\theta}_S = A(D_S)$.

It is usually computationally expensive to run the learning algorithm (i.e., a solver for the learning problem) over thousands of different input datasets. To reduce the computational workload of repeated training, we propose to learn and approximate the parameter function $A$ with deep learning models. That is, the optimized parameters are predicted through an evaluation of neural networks.

### 3.2 Algorithm

The proposed OptLEARN framework proceeds in two phases: offline training and online estimation, which are summarized in Figure 1.

**Offline Training**. The offline training of OptLEARN consists of the optimal parameters sampling step and the DNN training step. Denote $\pi$ as a permutation of all the training data points from the full set and $D_\pi^i$ is a set of points which precedes the $i$-th point in $\pi$. We adopt permutation sampling to sample the optimal parameters for DNN training. Denote the DNN model as $A_{DNN}$. The offline training workflow is summarized in Algorithm 1.

#### Algorithm 1: OptLEARN Offline Training

```plaintext
input : Full dataset $D = \{(x_i, y_i)\}_{i=1}^n$, learning algorithm $A$, the number of permutations $T$
output : Trained $A_{DNN}$
1 $\Phi \leftarrow \emptyset$
2 for $t = 1, \ldots, T$ do
3     $\pi_t \leftarrow \text{GenerateUniformRandomPermutation}(D)$
4     for $i = 1, \ldots, n$ do
5         $\hat{\theta}_i^{\pi_t} = A(D_\pi^i)$
6         $\Phi = \Phi \cup \{(D_\pi^i; \hat{\theta}_i^{\pi_t})\}$
7     end
8 end
9 return $A_{DNN}$ with $\Phi$
```

Compared with random sampling of subsets and optimal parameters, permutation sampling enables the evaluation of the change of model parameters with slightly varied subsets. Therefore, the training samples collected by permutation can better reflect the effect of individual data points, thus, allowing the DNN to better learn their contributions. The empirical experiment results also showed a slight advantage of permutation sampling than random sampling when we evaluated the performance of the DNN trained by subsets. Note that if the distribution of the subset to be estimated in the testing phase (i.e., test distribution) is known a priori, it is suggested to directly sample the training samples from the test distribution. For instance, in data valuation by SV, the definition of SV requires the distribution to assign uniform
With the set of training samples $\Phi = \{(D_1, \hat{\theta}_1), (D_2, \hat{\theta}_2), \ldots \}$, the objective of offline training phase is to train an effective parameter model which can predict the optimized parameters accurately for new subsets. For convex problems, Karuch-Kuhn-Tucker (KKT) conditions guarantee the optimality of a solution [Bertsekas, 1997]. Therefore, besides the direct training loss on predicted parameters: $L_{DNN} = \|A_{DNN}(D_S) - \hat{\theta}_S\|$, we consider the KKT loss to enforce the optimality of the predicted parameters. The utility loss is also added to maintain the similarity between the utility on the training subset $D_S$ calculated by predicted parameters and the ground-truth utility calculated by $\theta_S$.

For an unconstrained convex problem [1], the optimal parameters $\theta^*_S \in \arg\min_{\theta} L(\theta; D_S)$ should satisfy the stationary KKT condition as follows:

$$\nabla L(\theta^*_S; D_S) = 0.$$  \hspace{1cm} (2)

If the solver can find a near-optimal solution, then the stationary KKT condition is almost satisfied by the optimized parameter $\hat{\theta}_S$. Therefore, to compel the KKT condition, the KKT loss of one training sample is added as: $L_{KKT} = \|\nabla L(A_{DNN}(D_S); \hat{\theta}_S)\|$. It should be mentioned that for convex problems with hard constraints, we can still enforce the KKT stationary condition on the Lagrangian of the optimization problem.

For the utility loss, denote $\mathcal{U}(D_S; \hat{\theta})$ as the utility function that evaluates the utility of optimized model parameters $\hat{\theta}$ on a dataset $D_S$. For one training sample, $(\hat{\theta}_S; D_S)$, the utility loss is specified as: $L_U = |\mathcal{U}(D_S; A_{DNN}(D_S)) - \mathcal{U}(D_S; \hat{\theta}_S)|$. Thus, the total loss of one training sample for the proposed DNN is:

$$L_{DNN} = \|\hat{\theta}_S - A_{DNN}(D_S)\| + L_{KKT} + L_U. \hspace{1cm} (3)$$

**Online Estimation.** After the offline training phase, the trained DNN $A_{DNN}$ is used for efficient parameters prediction for a new training subset or a batch of subsets through an evaluation of $A_{DNN}$.

## 4 Characterization of Efficiently Approximatable Convex Optimizer

In this paper, we try to use a neural network to fit the function $\hat{\theta} = A(D)$. When the context is clear, we omit the learning algorithm and simply write the function as $\hat{\theta}(D)$. We denote the function associated with $k$th parameter as $\hat{\theta}_k(D)$, and thus $\hat{\theta}(D) = (\hat{\theta}_1(D), \ldots, \hat{\theta}_{d_{param}}(D))$. Although $D$ is a set of data points, it can be equivalently viewed as a vector concatenation of all $(x_i, y_i)$. We assume that we always normalize data features to $[0, 1]$. Therefore, we have $D \in [0, 1]^{n(d+1)}$. In order to fit $\hat{\theta}(D)$ with a deep neural network, the very first question is whether $\hat{\theta}(D)$ could be efficiently expressed or approximated by DNNs of certain structures. Here, the efficiency is measured by the total number of computational units in the DNNs. Despite the strong expressive power of DNNs, not every function could be efficiently approximated by them. Particularly, while the famous universal approximation theorem (UAT) includes many functions, there are requirements about continuity of the functions and compactness of the domain. Recent study [Yarotsky, 2017] shows that functions $f : \mathbb{R}^d \rightarrow \mathbb{R}$ that has a smaller upper bound of gradient norm $\|\nabla f\|$ could be more efficiently approximated by DNNs with ReLU activation. As long as $\theta_k(D)$ could be efficiently approximated by certain neural network architectures, we could put $d_{param}$ such networks in parallel to approximate $\hat{\theta}(D)$. The parallel networks share the same input but no internal connections and computational units. Therefore, in this section, we aim for understanding sufficient conditions for which $\|\nabla \hat{\theta}_k / D\|$ could be upper bounded.

Our first result is under the setting that the learning algorithm $A$ is able to find an optimal parameter $\theta^* \in \arg\min_{\theta} L(\theta; D)$. There are three major assumptions in our result: (1) the loss function $\ell(\theta)$ is $\alpha$-strongly convex, (2) $\left\|\frac{\partial}{\partial \theta} \ell(\theta, z)\right\|$ is upper bounded by some constant $B_1$, and (3) $\|\theta^*\| \leq B_2$.

**Definition 1** ($\alpha$-strongly convex). A differentiable function $f$ is $\alpha$-strongly convex if

$$f(y) \geq f(x) + \nabla f(x)^T(y - x) + \frac{\alpha}{2} \|y - x\|^2,$$  \hspace{1cm} (4)

for some $\alpha > 0$ and for all $x, y$ in the domain.

Since $\ell(\theta; (x, y))$ is $\alpha$-strongly convex, the training loss $L(D) = \frac{1}{n} \sum_{i=1}^n \ell(\theta; (x_i, y_i)) + \frac{\lambda}{2} \|\theta\|_2^2$ is $(\alpha + \lambda)$-strongly convex. Together with the condition that $\|\theta^*\|$ is finite, it implies $L$ has a unique global minimum $\theta^* = \arg\min_{\theta} L(\theta; D)$. Our main result for strongly convex functions is the following:
Theorem 1. If for all \( z \in [0, 1]^d \), the loss function \( \ell(\theta; z) \) is \( \alpha \)-strongly convex in \( \theta \), and \( \left\| \frac{\partial}{\partial \theta} \ell(\theta, z) \right\| \leq B_1 \) for \( i \in [d] \), then for \( \theta_k(D) = [\arg\min_\theta L(\theta; D)]_k \), where \( L(\theta; D) = \frac{1}{n} \sum_{i=1}^n \ell(\theta; (x_i, y_i)) + \frac{1}{2} \left\| \theta \right\|_2^2 \) and \( [.]_k \) means the \( k \)th element in the vector, then we have
\[
\left\| \frac{\partial \theta_k}{\partial D} \right\| \leq B_1 \frac{d_{\text{param}}(d+1)}{\sqrt{n}(\alpha + \lambda)}.
\] (5)

We then analyze the case where the learning algorithm \( \mathcal{A} \) is gradient descent. Gradient descent is one of the most commonly used optimization algorithms for convex learning problems. Formally, suppose the parameter \( \theta \) is initialized by 0 and denoted by \( \theta^{(0)} \). For the \( i \)th iteration, we update \( \theta^{(t)} = \theta^{(t-1)} - \eta \nabla L(\theta^{(t-1)}; D) \) with a learning rate \( \eta \). Given the maximum step number \( T \), we have \( \theta(D) = \theta^{(T)} \). The only different assumption for bounding \( \left\| \frac{\partial \theta_k}{\partial D} \right\| \) in this case is that the loss function \( \ell(\theta) \) is \( \beta \)-smooth instead of \( \alpha \)-strongly convex.

Definition 2 (\( \beta \)-smoothness). A differentiable function \( f \) is \( \beta \)-smooth if
\[
\|\nabla f(x) - \nabla f(y)\| \leq \beta \|x - y\| \tag{6}
\]
for some \( \beta > 0 \), and for all \( x, y \) in the domain.

We obtain the following result:

Theorem 2. If for all \( z \in [0, 1]^d \), the loss function \( \ell(\theta; z) \) is \( \beta \)-smooth in \( \theta \), and \( \left\| \frac{\partial}{\partial \theta} \ell(\theta, z) \right\| \leq B_1 \) for \( i \in [d] \), then for \( \theta_k^{(t)}(D) \) defined by the \( k \)th entry of \( \theta^{(t)} \), which is iteratively computed by \( \theta^{(t)} = \theta^{(t-1)} - \eta \nabla L(\theta^{(t-1)}; D) \) and \( \theta^{(0)} = 0 \) with a learning rate \( \eta > 0 \), and \( L(\theta; D) = \frac{1}{n} \sum_{i=1}^n \ell(\theta; (x_i, y_i)) + \frac{1}{2} \left\| \theta \right\|_2^2 \), we have
\[
\left\| \frac{\partial \theta_k^{(t)}}{\partial D} \right\| \leq (1 - (1 - \eta \lambda - \eta d_{\text{param}} \beta)^t) \frac{B_1 \sqrt{(d+1)}}{\sqrt{n}(\lambda + d_{\text{param}} \beta)}.
\]

5 Experiments

5.1 Experimental setup

Dataset. We evaluate the proposed OptLEARN on four datasets: Iris (Fisher and Creator [1988]) with input feature dimension \( d = 4 \), Spam (Hopkins et al. [1999], \( d = 215 \)), HIGGS (Baldi et al. [2014], \( d = 25 \)) and MNIST (LeCun et al. [1998], \( d = 128 \)). Using the binary version of Iris, we have three datasets (Iris, Spam, HIGGS) with two classes and one dataset (MNIST) with multiple classes. 300 training data points and 500 testing data points are randomly selected from Spam, HIGGS, and MNIST, and 2/3 points of the binary Iris are selected to generate the training sample set \( \Phi \), respectively. The preprocessing details can be found in the supplemental material.

Base model. To demonstrate that our proposed OptLEARN is a model-agnostic approach, two widely used convex models (i.e., Logistic Regression (LR) and Support Vector Machine (SVM)) are chosen as the base models from which OptLEARN will learn to refit. For both models, a regularization term with \( L_2 \)-norm coefficient \( \lambda = 1 \) is added to the loss function. We adopted L-BFGS-B [Zhu et al. [1997] as the solver for LR and LIBLINEAR [Fan et al. [2008] as the solver for SVM with a squared hinge loss.

Sampling Distribution. To train the OptLEARN model, 15000 subsets are sampled from each dataset following the permutation sampling procedures in Algorithm 1 to construct the training sample set \( \Phi \).

Evaluation Metrics and Baseline. To evaluate the performance of the proposed OptLEARN, baseline comparisons are conducted in two main scenarios: dataset deletion and dataset addition. During the deletion, the size of training subsets is decreased from 100% of the full size to 50% (i.e, for MNIST, from 300 to 150). During the addition, the size of training subsets is increased from 50% of the full size to 100% (i.e, for MNIST, from 150 to 300). In both scenarios, 10 subsets are generated randomly with each size. To illustrate the generalizability of the proposed OptLEARN to new data points from the same distribution, previously unseen data points are incorporated to form a larger subset in the dataset addition scenario.

In these two scenarios, influence function is selected as the baseline. With the ability to approximate the effect of single data point deletion or addition on model parameters, influence function can also be extended to evaluate the subset change [Koh et al. [2019]]. Besides, DeltaGrad [Wu et al. [2020b] can also serve as a baseline for batch deletion or addition. However, it requires the application of SGD to optimize the base model, which cannot provide the optimal
We visualize the result on MNIST in Figure 2. To assess the efficiency of OptLearn, we compare the performance of the optimal solution in a timely manner and cannot solve the primal problem of SVM accurately. Therefore, we do not include DeltaGrad in the experiments.

Three metrics are adopted to evaluate the effectiveness of the proposed method: 1) the Euclidean distance between the estimated model parameters and the exact parameters $\theta^*$ provided by the solver, i.e., $\|\hat{\theta} - \theta^*\|$; 2) the Normalized-Root-Mean-Squared Error (NRMSE) of the estimated utility calculated by the estimated model parameters; 3) the Spearman rank-order correlation (Myers and Sirois [2006]) between the ground-truth utility and the estimated utility on the testing set. The first metric evaluates the accuracy of parameter estimation. Here, we use the parameters optimized by retraining the model with the solver as the ground-truth. We adopt the Spearman correlation in addition to the NRMSE because most of the applications of subsets utility [Akhbardeh et al., 2019] desire that the utility be ranked in the correct order.

To assess the efficiency of OptLearn for the repeated training of the base model on a large number of subsets, we record the computation time required for influence function, OptLearn and the solver to update the model parameters, and calculated the average running time per subset.

**Machine configuration.** We run experiments with one Intel(R) Xeon(R) Gold 5218 CPU and use one GeForce RTX 2080 Ti for DNN training and inference.

**5.2 Experimental results.**

| Dataset | Algorithm | Parameter | Utility | Time (sec) | Parameter | Utility | Time (sec) |
|---------|-----------|-----------|---------|------------|-----------|---------|------------|
| Iris    | OptLearn  | 9.67E-02  | 4.23%   | 0.978E-01  | 5.11E-04  | 1.34E-01 | 5.10%      |
|         | Influence function | 8.97E-01 | 74.62% | 0.40711 | 7.64E-04 | 3.56E-01 | 63.07% | 0.0779 | 1.30E-04 |
|         | Exact solver | - | - | 3.90E-03 | - | - | - | 2.54E-03 |
| Spam   | OptLearn  | 3.72E-01  | 6.48%   | 0.985E-01  | 1.10E-03  | 5.96E-01 | 10.22%    |
|         | Influence function | 1.37E-00 | 50.00% | 0.1841 | 5.92E-02 | 1.71E-00 | 60.94% | 0.1886 | 2.10E-02 |
|         | Exact solver | - | - | 5.77E-03 | - | - | - | 6.24E-03 |
| HIGGS  | OptLearn  | 2.41E-01  | 12.58%  | 0.7980    | 5.43E-04  | 7.23E-01 | 20.66%    |
|         | Influence function | 3.32E-01 | 22.68% | 0.7504 | 1.12E-02 | 7.30E-01 | 17.34% | 0.7095 | 6.32E-03 |
|         | Exact solver | - | - | 5.13E-03 | - | - | - | 5.13E-03 |
| MNIST  | OptLearn  | 9.92E-01  | 2.28%   | 0.9978    | 1.09E-03  | 1.81E+00 | 5.46%     |
|         | Influence function | 1.04E+01 | 48.85% | 0.0278 | 1.93E-01 | 1.31E+01 | 125.26% | -0.8431 | 1.13E-01 |
|         | Exact solver | - | - | 3.14E-02 | - | - | - | 3.10E-02 |

Table 1: A summary of OptLearn results and baseline comparison results in the scenario of dataset deletion and addition with LR as the base model. The best result is highlighted in **bold**. The solver provides the optimal parameter for comparison, so the evaluation of its error is omitted as ‘-’.
which are formatted as the percentage of the size of full training set (i.e., $n = 300$ for MNIST). From Figure 2(c) and d, it is shown that influence function can make accurate approximation of the parameter change with a small scale of change (i.e., at the beginning of data addition and deletion). However, the distance grows with the scale of change rapidly due to nature of linear approximation of the influence function. In summary, OPTLEARN demonstrates its superior performance in predicting the model parameters with subsets of different sizes.

To evaluate the efficiency of OPTLEARN, we compute the average running time of estimating the parameter for each subset as the time recorded in Table 1. Note that this time does not include the computation time for offline training of OPTLEARN. Compared to the baseline and the solver, OPTLEARN demonstrates a significant advantage in efficiency since it only evaluates a deterministic NN to predict the model parameters. Given the offline training time, OPTLEARN gains its advantage in the application which requires the model retraining on a large number of subsets.

Despite the relative high computation cost during the offline training phase, the computation time of OPTLEARN grows much slower compared to the solver. It will be more efficient when the number of subsets to train exceeds roughly 40,000, which is a small number for applications such as SV estimation.

Due to space constraint, we defer results of two scenarios with SVM as the base model to Appendix, which demonstrate similar performance of OPTLEARN in terms of efficiency and effectiveness.

5.3 Applications

We investigate OPTLEARN performance in applications which require a large number of model retraining.

**Data Valuation.** Quantifying the value of each training data point to a learning task is a fundamental problem in ML. Shapley value is a widely used data value notion nowadays [Ghorbani and Zou, 2019; Jia et al., 2019a,b; Wang et al., 2020; Jia et al., 2021]. However, the exact SV calculation for a training dataset with $n$ points involves computing the marginal utility of every point in all subsets, which requires $2^n$ times of utility evaluation by retraining the model. Since OPTLEARN is capable of efficient utility evaluation by predicting the model parameters on a new training subset, OPTLEARN can speed up the SV calculation.

To validate the performance of OPTLEARN on calculating SV, we use permutation sampling [Maleki, 2015] to generate the subsets to be evaluated from the full training set $D$ in sequence. We compare the SV results calculated by
Learning to Refit for Convex Learning Problems

Table 2: A summary of OptLearn results and baseline comparison results in SV prediction with LR on IRIS, SPAM and HIGGS with permutation number \( T \in \{50, 100, 500, 1000\} \). The best result is highlighted in **bold**.

OptLearn, and UtlLearn (i.e., predicted SV) with the SV calculated by the solver as ground truth, and we also record the total time consumed by each method for subset utility evaluation. Here, we create UtlLearn as a baseline which has the same neural network structure as OptLearn but directly predicts the utility of a subset, and thus, not including KKT loss. We set the number of permutations \( T \in \{10, 50, 100, 500, 1000\} \) for Iris, Spam, and HIGGS. For MNIST, we set \( T \in \{10, 50, 100, 200\} \) due to the long computation time of the solver caused by high feature dimension. Similarly, we use the NRMSE and Spearman correlation as performance metrics to evaluate the predicted SV.

Table 2 summarizes the results for Iris, Spam and HIGGS. It can be observed that OptLearn outperforms UtlLearn in terms of SV prediction under all scenarios. Compared to UtlLearn, OptLearn has more network parameters to train due to the higher output dimension \( d_{\text{param}} \). However, OptLearn gains its advantage from the KKT regularizer which enforces the optimality of predicted parameters, thus, achieving accurate parameter and utility prediction.

Additionally, comparing the results of varied number of permutations, it is shown that the Spearman correlation tends to increase with number of permutations \( T \). When \( T \) is small, the ground-truth SV calculated by permutation sampling is very sensitive to the utility of small subsets. Therefore, the relative inaccurate prediction on small subsets results in the high NRMSE and low Spearman correlation. With the increase of \( T \), the SV predicted by OptLearn can better represent the true SV with the increasing correlation.

Figure 4 presents the trends of Spearman Correlation of predicted SV by OptLearn, and compares the total computation time of SV calculation with the solver on MNIST and HIGGS. It demonstrates the advantage of OptLearn over the solver in computation time, where time consumed by OptLearn has a slow growth with the increasing number of permutations.

Figure 4: Total computation time consumed by OptLearn and the solver on training different number of subsets in SV calculation for MNIST (left) and HIGGS (right).

**Model Calibration.** Classification models’ confidence calibration performance is crucial in mission-critical tasks [Guo et al., 2017], and a well-calibrated model should have a confidence (i.e., probability associated with the predicted class label) matching with its ground truth accuracy. Expected Calibration Error (ECE) [Naeini et al., 2015] is the primary metric of model calibration performance, which is calculated by partitioning predictions (with range \([0, 1]\)) into \( M \) equally-spaced bins and calculate the weighted average of differences between bins’ accuracy and confidence:

\[
ECE = \sum_{m=1}^{M} \frac{|B_m|}{N} |\text{acc}(B_m) - \text{conf}(B_m)|,
\]
where \( n \) is the number of training samples, \( B_m \) is the set of indices of samples whose prediction confidence falls into the interval \( I_m = (\frac{m-1}{M}, \frac{m}{M}) \) for \( m \in \{1, \ldots, M\} \). Accuracy and confidence are calculated below:

\[
\text{acc}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} 1[\hat{y}_i = y_i],
\]

\[
\text{conf}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \hat{p}_i,
\]

where \( \hat{y}_i \) and \( y_i \) are the predicted and true class labels of sample \( i \), respectively, and \( \hat{p}_i \) is the confidence for sample \( i \) on label \( y_i \). A model with better calibration performance should have a lower ECE.

### Table 3: Expected Calibration Error (ECE)

| Dataset | Regular \( \text{ECE} \) | OptLearn \( \text{ECE} \) |
|---------|-----------------|------------------|
| Iris    | 0.0562          | 0.0508 ± 0.0002  |
| SPAM    | 0.2093          | 0.1661 ± 0.0054  |
| HIGGS   | 0.0626          | 0.3082 ± 0.0151  |
| MNIST   | 0.5492          | 0.5227 ± 0.0014  |

One way to improve a model’s calibration performance is bagging [Breiman, 1996], which generates an ensemble of models on bootstrapped datasets and aggregates over each prediction to obtain final prediction results. However, training a large number of models can be time-consuming. We show here that OptLearn provides an efficient way to perform bagging, which effectively reduces the ECE. Specifically, we use LR base models for training the parameter function as a baseline of ensemble. Further, we combine these models with models predicted by the trained DNN, \( \mathcal{A}_{DNN} \), to get a larger ensemble. For generating models using \( \mathcal{A}_{DNN} \), each time we randomly sample a subset of instances with a ratio 0.6 from the training set with replacement and obtain estimated models predicted by \( \mathcal{A}_{DNN} \). As shown in Table 3, combining more models generated by \( \mathcal{A}_{DNN} \) effectively lower the ECE on Iris, Spam and MNIST dataset. ECE on HIGGS of OptLearn is higher than Regular, because the parameter function learned on HIGGS has a relative low performance, which may be caused by the low performance of the base model, LR, on this dataset, similar to the results of dataset deletion and addition.

### 6 Conclusion

Many essential applications in ML require retraining models on a large number of subsets of the same dataset, which has posed significant computational challenges. Existing methods are either incapable of estimating the large change on the dataset accurately or incapable of completing the repeated training efficiently. To address these challenges, we propose OptLearn to learn to refit models for convex learning problems by DNNs. The expressiveness of DNNs enables the effective approximation of the parameter function, and the proposed KKT and utility regularizers further improve the generalizability of OptLearn on subsets with a large change. We evaluate the effectiveness and efficiency of OptLearn on two dataset change scenarios and show that OptLearn outperforms state-of-the-art by providing accurate parameter estimation as well as achieving fast model retraining on a large number of subsets. The advantage of OptLearn is further demonstrated in two ML applications.

Several future work directions can be identified from this work. First, we will further explore the learnability of parameter function by DNNs. We would also like to generalize the proposed method by transferring the learned parameter function from one convex learning model to others, or even transferred to ensemble models [Li et al., 2017], which can further reduce the computation time.
References

Ruoxi Jia, David Dao, Boxin Wang, Frances Ann Hubis, Nezihe Merve Gürel, Bo Li, Ce Zhang, Costas J Spanos, and Dawn Song. Efficient task-specific data valuation for nearest neighbor algorithms. *arXiv preprint arXiv:1908.08619*, 2019a.

Ruoxi Jia, David Dao, Boxin Wang, Frances Ann Hubis, Nick Hynes, Nezihe Merve Gürel, Bo Li, Ce Zhang, Dawn Song, and Costas J Spanos. Towards efficient data valuation based on the shapley value. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1167–1176. PMLR, 2019b.

Ruoxi Jia, Xuehui Sun, Jiacen Xu, Ce Zhang, Bo Li, and Dawn Song. An empirical and comparative analysis of data valuation with scalable algorithms. *arXiv preprint arXiv:1911.07128*, 2019c.

Tianhao Wang, Johannes Rausch, Ce Zhang, Ruoxi Jia, and Dawn Song. A principled approach to data valuation for federated learning. In *Federated Learning*, pages 153–167. Springer, 2020.

Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning*, pages 2242–2251. PMLR, 2019.

Ryan Giordano, William Stephenson, Runjing Liu, Michael Jordan, and Tamara Broderick. A swiss army infinitesimal jackknife. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1139–1147. PMLR, 2019a.

Ryan Giordano, Michael I Jordan, and Tamara Broderick. A higher-order swiss army infinitesimal jackknife. *arXiv preprint arXiv:1907.12116*, 2019b.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *arXiv preprint arXiv:1612.01474*, 2016.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, pages 1885–1894. PMLR, 2017.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pages 3981–3989, 2016.

Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *International Conference on Machine Learning*, pages 748–756. PMLR, 2017.

Kaifeng Lv, Shunhua Jiang, and Jian Li. Learning gradient descent: Better generalization and longer horizons. In *International Conference on Machine Learning*, pages 2247–2255. PMLR, 2017.

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International Conference on Machine Learning*, pages 3751–3760. PMLR, 2017.

Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pages 4556–4565. PMLR, 2019.

Yue Cao, Tianlong Chen, Zhangyang Wang, and Yang Shen. Learning to optimize in swarms. *Advances in neural information processing systems*, 32:15044, 2019.

Xinshi Chen, Yu Li, Ramzan Umarov, Xin Gao, and Le Song. Rna secondary structure prediction by learning unrolled algorithms. *arXiv preprint arXiv:2002.05810*, 2020a.

Xinshi Chen, Hanjun Dai, Yu Li, Xin Gao, and Le Song. Learning to stop while learning to predict. In *International Conference on Machine Learning*, pages 1520–1530. PMLR, 2020b.

Antonio Ginart, Melody Y Guan, Gregory Valiant, and James Zou. Making ai forget you: Data deletion in machine learning. *arXiv preprint arXiv:1907.05012*, 2019.

Quoc Phong Nguyen, Bryan Kian Hsiang Low, and Patrick Jaillot. Variational bayesian unlearning. *Advances in Neural Information Processing Systems*, 33, 2020.

Jonathan Brophy and Daniel Lowd. Machine unlearning for random forests. In *International Conference on Machine Learning*, pages 1092–1104. PMLR, 2021.

Lucas Bourtoule, Varun Chandrasekaran, Christopher A Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine unlearning. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 141–159. IEEE, 2021.
Gert Cauwenberghs and Tomaso Poggio. Incremental and decremental support vector machine learning. *Advances in neural information processing systems*, pages 409–415, 2001.

Sebastian Schelter. amnesia—towards machine learning models that can forget user data very fast. In *1st International Workshop on Applied AI for Database Systems and Applications (AIDB19)*, 2019.

Yinjun Wu, Val Tannen, and Susan B Davidson. Priu: A provenance-based approach for incrementally updating regression models. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 447–462, 2020a.

Yinjun Wu, Edgar Dobriban, and Susan Davidson. Deltagrad: Rapid retraining of machine learning models. In *International Conference on Machine Learning*, pages 10355–10366. PMLR, 2020b.

Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal sunshine of the spotless net: Selective forgetting in deep networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9304–9312, 2020.

Samyadeep Basu, Philip Pope, and Soheil Feizi. Influence functions in deep learning are fragile. *arXiv preprint arXiv:2006.14651*, 2020.

Ananth Mahadevan and Michael Mathioudakis. Certifiable machine unlearning for linear models. *arXiv preprint arXiv:2106.15093*, 2021.

Diogo Almeida, Clemens Winter, Jie Tang, and Wojciech Zaremba. A generalizable approach to learning optimizers. *arXiv preprint arXiv:2106.00958*, 2021.

Dimitri P Bertsekas. Nonlinear programming. *Journal of the Operational Research Society*, 48(3):334–334, 1997.

Dmitry Yarotsky. Error bounds for approximations with deep relu networks. *Neural Networks*, 94:103–114, 2017.

R.A. Fisher and Test Creator. Iris. UCI Machine Learning Repository, 1988.

Mark Hopkins, Erik Reeber, George Forman, and Jaap Suermontd. Spambase. UCI Machine Learning Repository, 1999.

Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5(1):1–9, 2014.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4):550–560, 1997.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *the Journal of machine Learning research*, 9:1871–1874, 2008.

Pang Wei Koh, Kai-Siang Ang, Hubert HK Teo, and Percy Liang. On the accuracy of influence functions for measuring group effects. *arXiv preprint arXiv:1905.13289*, 2019.

Leann Myers and Maria J. Sirois. *Spearman Correlation Coefficients, Differences between*. American Cancer Society, 2006. ISBN 9780471667193. doi:https://doi.org/10.1002/0471667196.ess5050.pub2 URL https://onlinelibrary.wiley.com/doi/abs/10.1002/0471667196.ess5050.pub2

Alireza Akhbardeh, Hersh Sagreiya, Ahmed El Kaffas, Jürgen K Willmann, and Daniel L Rubin. A multi-model framework to estimate perfusion parameters using contrast-enhanced ultrasound imaging. *Medical physics*, 46(2):590–600, 2019.

Ruoxi Jia, Fan Wu, Xuehui Sun, Jiacen Xu, David Dao, Bhavya Kailkhura, Ce Zhang, Bo Li, and Dawn Song. Scalability vs. utility: Do we have to sacrifice one for the other in data importance quantification? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8239–8247, 2021.

Sasan Maleki. *Addressing the computational issues of the Shapley value with applications in the smart grid*. PhD thesis, University of Southampton, 2015.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017.

Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Yifu Li, Karuniya Mohan, Hongyue Sun, and Ran Jin. Ensemble modeling of in situ features for printed electronics manufacturing with in situ process control potential. *IEEE Robotics and Automation Letters*, 2(4):1864–1870, 2017.
A  Proof of Theorem 1

Theorem 1 (Restated). If for all \( z \in [0, 1]^d \), the loss function \( \ell(\theta; z) \) is \( \alpha \)-strongly convex in \( \theta \), and
\[
\left\| \frac{\partial}{\partial \theta z} \ell(\theta, z) \right\| \leq B_1
\]
for \( i \in [d] \), then for \( \theta_k(D) = \arg\min_{\theta} L(\theta; D) \), where \( L(\theta; D) = \frac{1}{n} \sum_{i=1}^{n} \ell(\theta; (x_i, y_i)) + \frac{\lambda}{2} \| \theta \|^2 \) and \( [\cdot]_k \) means the \( k \)th element in the vector, then we have
\[
\left\| \frac{\partial \theta_k}{\partial D} \right\| \leq B_1 \sqrt{\frac{d_{\text{param}} (d+1)}{n (\alpha + \lambda)}}.
\]

Proof. Given a loss function \( L(\theta; D) = \frac{1}{n} \sum_{i=1}^{n} \ell(\theta; (x_i, y_i)) + \frac{\lambda}{2} \| \theta \|^2 \), the unique local minimum \( \hat{\theta}(D) \) is characterized by the implicit function
\[
\nabla L(\hat{\theta}, D) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\theta} \ell(\hat{\theta}; (x_i, y_i)) + \lambda \hat{\theta} = 0.
\]

Denote \( \sigma(j) = [j/(d+1)] \) to be the index of the data point (and hence the loss function \( \ell \) \( j \)th dimension of \( D \) corresponds to). By the implicit function theorem and the chain rule, we have
\[
\left( \sum_{i=1}^{n} \frac{\partial \ell_i}{\partial \theta^T \theta} + n \lambda I \right) \frac{\partial \theta}{\partial D} + \frac{\partial \ell_{\sigma(j)}}{\partial \theta \partial D_j} = 0,
\]
where \( \ell_i = \ell(\cdot; (x_i, y_i)) \) and, therefore, we obtain
\[
\frac{\partial \theta}{\partial D_j} = -H^{-1} \frac{\partial \ell_{\sigma(j)}}{\partial \theta \partial D_j},
\]
where \( H = \sum_{i=1}^{n} \frac{\partial \ell_i}{\partial \theta^T \theta} + n \lambda I \) is the Hessian matrix. We use \( [H^{-1}]_k \) to denote the \( k \)th row of the Hessian inverse, and, hence,
\[
\frac{\partial \theta}{\partial D_j} = \left[ \frac{\partial \theta_1}{\partial D_j}, \ldots, \frac{\partial \theta_{d_{\text{param}}}}{\partial D_j} \right] = \left[ -[H^{-1}]_1 \frac{\partial \ell_{\sigma(j)}}{\partial \theta \partial D_j}, \ldots, -[H^{-1}]_{d_{\text{param}}} \frac{\partial \ell_{\sigma(j)}}{\partial \theta \partial D_j} \right].
\]

Therefore, we have
\[
\left\| \frac{\partial \theta}{\partial D_j} \right\|_2^2 = \sum_{k=1}^{d_{\text{param}}} \left( [H^{-1}]_k \frac{\partial \ell_{\sigma(j)}}{\partial \theta \partial D_j} \right)^2 \leq \sum_{k=1}^{d_{\text{param}}} \left\| [H^{-1}]_k \right\|^2 \left\| \frac{\partial \ell_{\sigma(j)}}{\partial \theta \partial D_j} \right\|^2 \leq \left\| \frac{\partial \ell_{\sigma(j)}}{\partial \theta \partial D_j} \right\|^2 \left\| H^{-1} \right\|_F^2 \leq B_1^2 \left\| H^{-1} \right\|_F^2,
\]
where the inequality is due to the Cauchy–Schwarz inequality.

The remaining work is to bound \( \left\| H^{-1} \right\|_F \). Denote \( \lambda_1, \ldots, \lambda_{d_{\text{param}}} \) as the eigenvalues of \( H \). Observe that
where the second equality holds due to \( H \) being symmetric, hence \( H^{-1} \) is also symmetric. Further, the last equality holds, because the eigenvalues for \( H^{-1} \) are \( \frac{1}{\lambda_1}, \ldots, \frac{1}{\lambda_{d_{\text{param}}}} \). Since each \( \ell \) is \( \alpha \)-strongly convex, we know that the minimum eigenvalue of the Hessian matrix \( H \) is at least \( n\alpha + n\lambda \). Therefore, we obtain
\[
\|H^{-1}\|_F \leq \frac{\sqrt{d_{\text{param}}}}{n\alpha + n\lambda}.
\]

Then, it follows that
\[
\left\| \frac{\partial \theta}{\partial D_j} \right\|_2 \leq B_1 \frac{\sqrt{d_{\text{param}}}}{n(\alpha + \lambda)},
\]
which implies
\[
\left| \frac{\partial \theta_k}{\partial D_j} \right| \leq B_1 \frac{\sqrt{d_{\text{param}}}}{n(\alpha + \lambda)},
\]
and, finally, we have
\[
\left| \frac{\partial \theta_k}{\partial D} \right| \leq B_1 \frac{\sqrt{d_{\text{param}}}(d+1)}{\sqrt{n}(\alpha + \lambda)}.
\]

\[
\text{(18)} \\
\text{(19)} \\
\text{(20)} \\
\text{(21)} \\
\text{(22)} \\
\text{(23)} \\
\text{B Proof of Theorem 2}
\]

**Theorem 2.** If for all \( z \in [0, 1]^d \), the loss function \( \ell(\theta; z) \) is \( \beta \)-smooth in \( \theta \), and \( \left| \frac{\partial}{\partial \theta} \ell(\theta, z) \right| \leq B_1 \) for \( i \in [d] \), then for \( \theta^{(t)}(D) \) defined by the \( k \)th entry of \( \theta^{(t)} \), which is iteratively computed by \( \theta^{(t)} = \theta^{(t-1)} - \eta \nabla L(\theta^{(t-1)}; D) \) and \( \theta^{(0)} = 0 \) with a learning rate \( \eta > 0 \), and \( L(\theta; D) = \frac{1}{n} \sum_{i=1}^{n} \ell(\theta; (x_i, y_i)) + \frac{1}{2} \|\theta\|_2^2 \), we have
\[
\left( 1 - (1 - \eta \lambda - \eta d_{\text{param}}) B_1 \right) \frac{\sqrt{d+1}}{\sqrt{n}(\lambda + d_{\text{param}})}.
\]

**Proof.** Consider an iteration \( \theta_k^{(t)} = \theta_k^{(t-1)} - \eta \nabla L(\theta^{(t-1)}; D) \). Taking a derivative with respect to \( D_j \) for both sides, we have
\[
\frac{\partial \theta_k^{(t)}}{\partial D_j} = \frac{\partial \theta_k^{(t-1)}}{\partial D_j} - \eta \left[ \lambda \frac{\partial \theta_k^{(t-1)}}{\partial D_j} + \frac{1}{n} \sum_{i=1}^{n} \frac{\partial^2 \ell_i}{\partial \theta_k^2} \frac{\partial \theta_k^{(t-1)}}{\partial D_j} + \frac{1}{n} \frac{\partial \ell_{\sigma(j)}}{\partial \theta_k} \frac{\partial \theta_k^{(t-1)}}{\partial D_j} \right]
\]
\[
\text{(24)} \\
\text{(25)} \\
\text{(26)} \\
\text{where the inequality follows due to } \left| \frac{\partial \ell_{\sigma(j)}}{\partial \theta_k} \right| \leq B_1 \text{ and } \left| \frac{\partial^2 \ell_i}{\partial \theta_k^2} \right| \leq d_{\text{param}}. \]
Observe that

\[
\frac{\partial \theta^{(t)}}{\partial D_j} + \eta B_1 \leq (1 - \alpha) \left( \frac{\partial \theta^{(t-1)}}{\partial D_j} + \eta B_1 \right)
\]

(27)

\[
\leq (1 - \alpha)^t \eta B_1.
\]

(28)

Therefore, we obtain

\[
\left| \frac{\partial \theta^{(t)}}{\partial D_j} \right| \leq (1 - (1 - \eta \lambda - \eta d_{param} \beta)^t) \frac{B_1}{n(\lambda + d_{param} \beta)}.
\]

(29)

which results in

\[
\left\| \frac{\partial \theta^{(t)}}{\partial D} \right\| \leq (1 - (1 - \eta \lambda - \eta d_{param} \beta)^t) \frac{B_1 \sqrt{(d + 1)}}{\sqrt{n}(\lambda + d_{param} \beta)}.
\]

(30)

\[\square\]

### C Expressiveness of Deep ReLU Network

Yarotsky [2017] shows that continuous differentiable functions with a smaller upper bound of the gradient norm can be more efficiently approximated by deep ReLU networks. For completeness, we present the result here.

**Theorem 3.** For any Lipschitz continuous function \( f : [0,1]^d \to \mathbb{R} \) with range \( B \) and \( \left\| \frac{\partial f}{\partial x} \right\| \leq L \) for all \( x \in [0,1]^d \), there is a ReLU network that is capable of expressing any such function within an arbitrary error \( \varepsilon \) and has no more than \( c \ln(2^{d+1} B d (d + 1)/\varepsilon) + 1 \) layers and \( d \left( 2^{d+1} d L / \varepsilon + 1 \right)^2 (c \ln(2^{d+1} B d (d + 1)/\varepsilon) + 1) \) computational units.

As we can see, if \( L \) is smaller, the upper bound of the network size is smaller. The proof of this theorem is done by simply plugging in constants for Theorem 3.1 in Yarotsky [2017].

### D EXPERIMENTS

#### D.1 Datasets

**Iris** (Fisher and Creator [1988]). We use the binary version of Iris dataset with the first two classes of data. The binary version contains 100 samples in total with feature dimension \( d = 4 \). 67 data points are randomly selected to generate training subsets. The remaining 33 data points are used for Shapley value calculation, and 17 of them are used for dataset addition.

**Spam** (Hopkins et al. [1999]). Spam dataset is a collection of spam and non-spam e-mails with feature dimension \( d = 215 \). 300 data points are randomly selected to generate training subsets. 500 data points are used as the testing set for Shapley value calculation, and 150 of them are used for dataset addition.

**HIGGS** (Baldi et al. [2014]). HIGGS dataset is produced using Monte Carlo simulations with feature dimension \( d = 30 \). After preprocessing, we keep 25 features. 300 data points are randomly selected to generate training subsets. 500 data points are used as the testing set for Shapley value calculation, and 150 of them are used for dataset addition.

**MNIST** (LeCun et al. [1998]). MNIST dataset is a collection of grayscale handwritten digits with size \( 28 \times 28 \) and 10 classes. We use a CNN in the preprocessing stage to extract the features and reduce the dimension to 128. 300 data points are randomly selected to generate training subsets. 500 data points are used as the testing set for Shapley value calculation, and 150 of them are used for dataset addition.

All the input data \( x \) are normalized to rescale the norm \( \| x \| \) to be within the range of \([0, 1]\)

#### D.2 Setup

Table 4 details the size of the added or deleted subsets for each dataset for two main scenarios in the experiment: dataset deletion and dataset addition.

\[\text{Table 4}\]
### D.3 Additional Results

We rerun the experiments to record the running time of each method and calculate the standard deviation (Std). We also add Std for all other evaluation metrics for both dataset deletion and dataset addition experiments with Logistic Regression (LR) and Support Vector Machine (SVM) as base models, which are summarized in Table 5, 6, 7 and 8.

Additionally, we do the ablation study through creating PARALEASE as another baseline. PARALEASE has the same neural network structure as OPTLEARN and also predicts the optimized parameter for a convex learning model. The difference is that it does not include the KKT regularizer and utility regularizer. Its performance is examined in dataset deletion and addition scenarios with LR as the base model and summarized in Table 5 and 6.

From Table 5 and 6, it can be shown that although PARALEASE outperforms the Influence Function on Iris dataset, the Spearman correlation for other datasets does not exist. This is because it provides constant prediction regardless of the input subset change, resulting in constant utility for all subsets. Therefore, this result reflects the effectiveness of the input subset change.

---

**Table 5: A summary of OPTLEARN results and baseline comparison results in the scenario of dataset deletion with LR as the base model. The best result is highlighted in bold.**

| Dataset | Algorithm | Parameter | Utility | Avg Running Time (sec) |
|---------|-----------|-----------|---------|------------------------|
|         |           | $\theta - \theta^*$ | Std | NRMSE Std | Spearman Corr Std | Time Std |
| Iris    | OPTLEARN  | Influence function | 9.67E-02 | 8.1E-03 | 4.23% | 1.62% | 0.9786 | 0.0037 | 1.41E-03 | 1.43E-03 |
|         |           | Exact solver | 8.97E-01 | 1.39E-02 | 74.62% | 16.98% | 0.4071 | 0.4262 | 1.41E-03 | 6.75E-04 |
|         | PARALEARN | Influence function | 1.95E-01 | 2.49E-02 | 30.53% | 10.74% | 0.6821 | 0.2538 | 5.34E-04 | 6.64E-05 |
|         |           | Exact solver | - | - | - | - | - | - | 5.68E-03 | 3.65E-03 |
| Spam    | OPTLEARN  | Influence function | 1.72E-01 | 1.27E-02 | 6.48% | 1.19% | 0.9856 | 0.0040 | 1.52E-03 | 1.28E-03 |
|         |           | Exact solver | 1.07E+00 | 3.40E-03 | 50.60% | 3.70% | 0.1841 | 0.1797 | 4.19E-02 | 1.63E-03 |
|         | PARALEARN | Influence function | 2.36E+00 | 2.15E-02 | 125.55% | 10.62% | N/A | N/A | 1.51E-03 | 1.23E-03 |
|         |           | Exact solver | - | - | - | - | - | - | 1.60E-02 | 1.08E-02 |
| HIGGS   | OPTLEARN  | Influence function | 2.41E-01 | 1.77E-02 | 12.58% | 3.86% | 0.7980 | 0.0662 | 6.60E-04 | 1.09E-05 |
|         |           | Exact solver | 3.32E-01 | 5.11E-03 | 22.68% | 6.11% | 0.7504 | 0.1275 | 1.45E-02 | 8.17E-04 |
|         | PARALEARN | Influence function | 5.11E-01 | 2.89E-02 | 65.30% | 17.03% | N/A | N/A | 6.61E-04 | 1.63E-05 |
|         |           | Exact solver | - | - | - | - | - | - | 5.26E-03 | 5.33E-04 |
| MNIST   | OPTLEARN  | Influence function | 9.92E-01 | 9.60E-03 | 2.28% | 0.42% | 0.9978 | 0.0010 | 1.16E-03 | 4.70E-05 |
|         |           | Exact solver | 1.04E+00 | 6.65E-01 | 48.85% | 2.81% | 0.0278 | 0.2375 | 1.73E-01 | 2.78E-17 |
|         | PARALEARN | Influence function | 4.58E+00 | 4.96E-03 | 113.94% | 7.11% | N/A | N/A | 1.05E-03 | 4.13E-05 |
|         |           | Exact solver | - | - | - | - | - | - | 4.13E-02 | 8.71E-03 |

---

**Table 6: A summary of OPTLEARN results and baseline comparison results in the scenario of dataset addition with LR as the base model. The best result is highlighted in bold.**

| Dataset | Algorithm | Parameter | Utility | Avg Running Time (sec) |
|---------|-----------|-----------|---------|------------------------|
|         | $\theta - \theta^*$ | Std | NRMSE Std | Spearman Corr Std | Time Std |
| Iris    | OPTLEARN  | Influence function | 1.31E-01 | 8.1E-03 | 5.10% | 0.52% | 0.9882 | 0.0067 | 1.65E-03 | 1.76E-03 |
|         |           | Exact solver | 3.56E-01 | 6.13E-04 | 63.07% | 3.15% | 0.0779 | 0.1521 | 3.09E-04 | 1.47E-04 |
|         | PARALEARN | Influence function | 2.85E-01 | 4.22E-03 | 43.91% | 2.83% | 0.9730 | 0.0161 | 2.31E-03 | 2.27E-03 |
|         |           | Exact solver | - | - | - | - | - | - | 4.43E-03 | 9.72E-04 |
| Spam    | OPTLEARN  | Influence function | 5.96E-01 | 4.66E-03 | 10.22% | 0.82% | 0.9919 | 0.0026 | 1.75E-03 | 1.98E-03 |
|         |           | Exact solver | 1.11E+00 | 2.34E-03 | 65.44% | 2.26% | 0.1886 | 0.1683 | 2.12E-02 | 1.42E-03 |
|         | PARALEARN | Influence function | 2.13E+00 | 2.10E-02 | 145.73% | 4.36% | N/A | N/A | 3.91E-03 | 6.67E-03 |
|         |           | Exact solver | - | - | - | - | - | - | 1.17E-02 | 8.31E-03 |
| HIGGS   | OPTLEARN  | Influence function | 3.26E-01 | 1.56E-02 | 14.11% | 2.71% | 0.8054 | 0.0684 | 3.62E-03 | 5.57E-03 |
|         |           | Exact solver | 4.58E-01 | 3.23E-03 | 20.40% | 4.27% | 0.9008 | 0.0381 | 8.35E-03 | 3.07E-03 |
|         | PARALEARN | Influence function | 1.92E+00 | 1.86E-02 | 838.62% | 128.59% | N/A | N/A | 3.65E-03 | 6.66E-03 |
|         |           | Exact solver | - | - | - | - | - | - | 8.28E-03 | 4.89E-03 |
| MNIST   | OPTLEARN  | Influence function | 1.81E-01 | 1.22E-02 | 5.46% | 0.25% | 0.9960 | 0.0009 | 1.00E-03 | 3.34E-05 |
|         |           | Exact solver | 1.31E+01 | 4.75E-01 | 125.26% | 4.22% | -0.8431 | 0.0367 | 1.73E-01 | 7.60E-03 |
|         | PARALEARN | Influence function | 4.76E+00 | 6.01E-03 | 131.73% | 2.17% | N/A | N/A | 8.95E-04 | 3.80E-05 |
|         |           | Exact solver | - | - | - | - | - | - | 4.20E-02 | 3.64E-03 |
| Dataset  | Algorithm          | Parameter | Utility          | Avg Running Time (sec) |
|---------|--------------------|-----------|------------------|------------------------|
|         |                    | $\|\hat{\theta} - \theta^*\|$ | NRMSE | Std | Spearman Corr | Std | Time | Std |
| Iris    | OptLearn Influence function | 1.16E-01 | 1.53E-02 | 17.25% | 7.70% | 0.9214 | 0.0763 | 5.62E-04 | 6.51E-05 |
|         | Exact solver       | 4.63E-01 | 8.31E-03 | 54.10% | 8.99% | 0.0143 | 0.4427 | 1.61E-03 | 5.48E-04 |
|         |                    | 8.74E-01 | 1.38E-02 | 6.85%  | 0.55% | 0.9826 | 0.0059 | 1.34E-03 | 9.98E-04 |
|         | Influence function | 1.47E+00 | 7.78E-03 | 48.96% | 3.23% | 0.1395 | 0.2271 | 5.43E-02 | 2.92E-03 |
|         | Exact solver       | 6.12E-01 | 6.61E-03 | 16.97% | 2.78% | 0.7274 | 0.1377 | 2.27E-03 | 3.93E-04 |
| Spam    | OptLearn Influence function | 5.38E-01 | 2.72E-02 | 7.52%  | 1.16% | 0.9048 | 0.0442 | 5.56E-04 | 1.70E-05 |
|         | Exact solver       | 1.19E+00 | 1.77E-02 | 10.44% | 1.04% | 0.9848 | 0.0464 | 1.45E-03 | 8.20E-04 |
|         | Influence function | 1.62E+00 | 3.27E-03 | 55.03% | 2.01% | 0.8080 | 0.0638 | 2.96E-02 | 2.69E-03 |
|         | Exact solver       | 7.23E-01 | 2.07E-02 | 20.66% | 2.55% | 0.8467 | 0.0378 | 2.36E-03 | 5.76E-04 |
| HIGGS   | OptLearn Influence function | 7.30E-01 | 7.37E-03 | 17.34% | 2.41% | 0.7095 | 0.0828 | 8.96E-03 | 5.97E-04 |

Table 7: A summary of OPTLEARN results and baseline comparison results in the scenario of dataset deletion with SVM as the base model. The best results are highlighted in **bold**.

| Dataset  | Algorithm          | Parameter | Utility          | Avg Running Time (sec) |
|---------|--------------------|-----------|------------------|------------------------|
|         |                    | $\|\hat{\theta} - \theta^*\|$ | NRMSE | Std | Spearman Corr | Std | Time | Std |
| Iris    | OptLearn Influence function | 1.68E-01 | 1.15E-02 | 40.82% | 3.01% | 0.9292 | 0.0379 | 7.19E-04 | 3.98E-05 |
|         | Exact solver       | 1.50E-01 | 6.72E-04 | 63.29% | 3.88% | 0.1694 | 0.2300 | 1.78E-04 | 6.95E-04 |
|         | Influence function | 1.19E+00 | 1.77E-02 | 10.44% | 1.04% | 0.9848 | 0.0464 | 1.45E-03 | 8.20E-04 |
|         | Exact solver       | 1.62E+00 | 3.27E-03 | 55.03% | 2.01% | 0.8080 | 0.0638 | 2.96E-02 | 2.69E-03 |
|         | Influence function | 7.23E-01 | 2.07E-02 | 20.66% | 2.55% | 0.8467 | 0.0378 | 8.96E-03 | 5.97E-04 |
|         | Exact solver       | 7.30E-01 | 7.37E-03 | 17.34% | 2.41% | 0.7095 | 0.0828 | 2.21E-03 | 8.63E-04 |

Table 8: A summary of OPTLEARN results and baseline comparison results in the scenario of dataset addition with SVM as the base model. The best results are highlighted in **bold**.

of the proposed KKT and utility loss, which enforce the optimality of predicted parameters and greatly enhance the generalization ability of the DNN.

The results in Tables 7 and 8 illustrate that the proposed OPTLEARN achieves similar performance with different convex learning problems (i.e., LR and SVM) in terms of the accuracy of parameter prediction and strong correlation between the predicted utility and the actual utility of subsets.