EFFICIENT SECOND-ORDER OPTIMIZATION FOR DEEP LEARNING WITH KERNEL MACHINES

Anonymous authors
Paper under double-blind review

ABSTRACT

Second-order optimization has been recently explored in neural network training. However, the recomputation of the Hessian matrix in the second-order optimization posts much extra computation and memory burden in the training. There have been some attempts to address the issue by approximation on the Hessian matrix, which unfortunately degrades the performance of the neural models. To address the issue, we propose Kernel Stochastic Gradient Descent (Kernel SGD) which projects the optimization problem to a transformed space with the Hessian matrix of kernel machines. Kernel SGD eliminates the recomputation of the Hessian matrix and requires a much smaller memory cost which can be controlled via the mini-batch size. The additional advantage of Kernel SGD is its ability to converge to better solutions according to our theoretical analysis. Kernel SGD is theoretically guaranteed to converge. Experimental results on tabular, image and text data show that Kernel SGD converges up to 30 times faster than the existing second-order optimization techniques, and also shows remarkable performance in generalization.

1 INTRODUCTION

A recent trend in training neural networks is to exploit the second-order information to escape the saddle points and converge to a better optimum (Dauphin et al., 2015; Yao et al., 2018). Such methods use the second-order derivatives on the weights (i.e., edges or connections) of the neural networks, and the second-order derivatives together form a Hessian matrix for each layer. The extensive computation of the Hessian matrix posts much extra computation and memory burden in the training, as the Hessian matrix which is computed based on the neural network weights needs to be updated in each iteration of the training. The computation and memory cost become huge for networks with a large number of weights, as the size of the Hessian matrix is quadratic in the number of weights. Researchers attempted to improve the efficiency and memory cost of the second-order optimization with approximation such as quasi-Newton (Liu & Nocedal, 1989), Fisher information (Martens & Grosse, 2015) and diagonalization (Dauphin et al., 2015). Nonetheless, the approximation may lead to the degradation on the model performance such as the predictive accuracy.

Kernel machines (e.g., SVMs) have achieved comparable performance with neural networks when solving some machine learning problems. For example, a recent study (Wen et al., 2021) on a popular sentiment analysis problem shows that SVM based solutions can achieve competitive predictive accuracy to the deep neural network based approaches (Devlin et al., 2018). Belkin et al. (2018) demonstrated that kernel machines can fit the problem with random labels easily and produce robust generalization comparable to neural networks. As kernel machines take advantage of convexity and have used second-order optimizations in the training (Hao et al., 2016), incorporation of the second-order information from convex kernel machine problems into non-convex deep learning problems may guide the optimization to a better direction. Based on this inspiration, we exploit the second-order information from kernel machines and propose a Kernel Stochastic Gradient Descent (hereafter "Kernel SGD") optimization method. The Hessian matrix in Kernel SGD is proportional to the size of the mini-batch of training instances, rather than the number of weights. As a result, the size of the Hessian matrix in Kernel SGD can be controlled by setting the size of the mini-batch. Another important property of Kernel SGD is that the Hessian matrix does not need to be updated in each iteration, because it is computed based on the training instances which are unchanged during backward and forward propagation. Kernel SGD uses the Hessian matrix of kernel machines

to project the original neural network problem to another space, where the optimum tends to be closer to the initial point according to our theoretical analysis. Thus Kernel SGD is more likely to converge to a better solution. We further provide the theoretical guarantee on the convergence of the neural network training using Kernel SGD. Experimental results show that Kernel SGD converges up to 30 times faster, and achieves a remarkable generalization, in comparison with existing second-order optimization. The memory cost for Hessian matrix in Kernel SGD is much smaller especially with larger neural networks. As a sanity check, we also compare Kernel SGD with the first-order optimization. Our results show that Kernel SGD even outperforms the first-order optimization in terms of generalization and convergence time in some problems tested. To summarize, our main contributions in this paper are listed below.

- We propose Kernel SGD which incorporates the second-order information of kernel machines into the training of neural networks. Kernel SGD exploits the Hessian matrix of the kernel machine which is proportional to the size of mini-batch, and the size of which is controllable by practitioners. The recomputation of Hessian matrix is eliminated benefiting from the unchanged Hessian matrix.
- We theoretically prove that the optimization using Kernel SGD is guaranteed to converge and our theoretical analysis indicates that Kernel SGD is more likely to converge fast to a better solution.
- We conduct extensive experiments on tabular, image and text data to investigate the behaviours of Kernel SGD. The experimental results show that Kernel SGD converges up to 30 times faster while performs a remarkable generalization, in comparison with the second-order optimization baselines. Kernel SGD even outperforms the first-order optimization in some circumstances.

2 Preliminaries on Kernel Machines

A kernel machine uses the kernel trick to map the non-linear problem into a feature space where the problem may be linearly separable with an appropriate kernel function (Keerthi & Lin, 2003; Hofmann et al., 2008). Next we give the formal definition of kernel machines discussed in this paper. Given a training data set $\{\boldsymbol{X}, \boldsymbol{y}\}$ of n training instances where $\{\boldsymbol{X} \in \mathbb{R}^{n\times d}, \boldsymbol{y} \in \mathbb{R}^n\} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_n, y_n)\}$, and $(\boldsymbol{x}_i, y_i)$ denotes the instance $\boldsymbol{x}_i \in \mathbb{R}^d$ with its label y_i, the objective of the kernel machine training is to find an optimal $\boldsymbol{\omega}^*$ which minimizes the structural risk as follows.

$$\min L(\boldsymbol{\omega}) = \frac{1}{n}\sum_{i=1}^{n} l(f(\boldsymbol{\omega}, \boldsymbol{x}_i), y_i) + \frac{\lambda}{2}||\boldsymbol{\omega}||^2, \tag{1}$$

where λ denotes the regularization constant and $f(\boldsymbol{\omega}, \boldsymbol{x}_i) = \langle \boldsymbol{\omega}, \phi(\boldsymbol{x}_i)\rangle$. The variable $\boldsymbol{\omega}$ is defined on the *reproducing kernel Hilbert space* (RKHS) and $\langle \cdot, \cdot \rangle$ is the inner product on the RKHS. The function $\phi(\cdot)$ maps the instances from their original data space to a higher dimensional feature space induced by the kernel function. Assume the loss $l(\cdot, \cdot)$ is an affine function of $\boldsymbol{\omega}$. The *representer theorem* (Schölkopf et al., 2001) shows that a minimizer of the optimization problem (1) is $\boldsymbol{\omega} = \sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j)$. Based on the *reproducing property* (Smola & Schölkopf, 1998), we have $f(\boldsymbol{\omega}, \boldsymbol{x}_i) = \sum_{j=1}^{n} \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ where $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ denotes a positive definite kernel function and $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j)\rangle$. By substituting the expressions of $f(\boldsymbol{\omega}, \boldsymbol{x}_i)$ and $\boldsymbol{\omega}$ into the Equation (1), we have the objective with respect to $\boldsymbol{\alpha}$ below.

$$\min L(\boldsymbol{\alpha}) = \frac{1}{n}\sum_{i=1}^{n} l(\sum_{j=1}^{n} \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j), y_i) + \frac{\lambda}{2}||\sum_{j=1}^{n} \alpha_j \phi(\boldsymbol{x}_j)||^2, \tag{2}$$

where $\boldsymbol{\alpha} = [\alpha_1 \ \dots \ \alpha_n]^T$ is an n-dimension vector, each dimension of which corresponds to the contribution of a training instance to the kernel machine.

Then, we can derive that the Hessian matrix $H = [H_{ij}]_{n\times n}$ of Problem (2) is equal to the kernel matrix. The element in the i-th row and j-th column of the matrix H is $H_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The derivations of the first and second derivatives can be found in the supplementary material. For clarity, we use kernel matrix H to denote the Hessian matrix of the kernel machine in the rest of the paper.

3 OUR PROPOSED KERNEL SGD OPTIMIZATION

In this section, we aim to elaborate our techniques to improve the efficiency of the second-order optimization in neural network training. There are two challenges in exploiting second-order information. First, the memory and computation cost of second-order optimization is considerably high, due to the large size of the weight matrix and the frequent update of the Hessian matrix in the training of the neural network. Second, simple methods like approximation on the Hessian matrix, although show improvements on memory and computation, may lead to the deficiency in the model performance such as the predictive accuracy.

In order to tackle these challenges, we take advantages of kernel machines and propose an optimization method named "Kernel Stochastic Gradient Descent" or Kernel SGD for short. Kernel machines can solve some problems as good as neural network (Belkin et al., 2018; Wen et al., 2021), but conduct the convex optimization. Inspired by the recent findings, our method integrates the second-order information of the kernel machine into the neural network optimization, which may help escape the saddle points and accelerate the convergence in the optimization of non-convex problems. The key idea of Kernel SGD is to project the optimization problem to a transformed space with the kernel matrix of kernel machines, where the optimum tends to be closer to the starting point based on our theoretical analysis. Thus Kernel SGD is more likely to converge to a better solution. Considering the difference between kernel machines and neural models, we tailor the kernel matrix to the optimization of neural networks, with theoretical guarantees on convergence. By exploiting the kernel matrix from kernel machines, our method shows advantages in computation and space efficiency. The size of the kernel matrix is related to the size of input batch which is much smaller than that in the original neural network problem. The kernel matrix of the kernel machine stays unchanged during the whole training, which can significantly save the training time. Next, we provide details in the projection process and the update rule of Kernel SGD with theoretical analysis.

3.1 PROBLEM PROJECTION AND UPDATE RULE OF KERNEL SGD

Kernel SGD uses the the second-order information in kernel machines to project the neural network problem into another space in which gradient descent may converge to a better solution. Formally, to project the original optimization objective $\mathcal{J}(W)$ (e.g., cross entropy loss) of the neural network, the weight matrix $W \in \mathbb{R}^{d_{out} \times d_{in}}$ is first projected to a new matrix called $\hat{W}$. By introducing the projection matrix P, we have the projected weight $\hat{W} = WP^{\frac{1}{2}}$ where $P \in \mathbb{R}^{d_{in} \times d_{in}}$ is a symmetric matrix. The projection matrix is used to project the optimization problem into another space. Thus the problem after projection becomes optimizing the projected loss $\hat{\mathcal{J}}(\hat{W})$ where $\mathcal{J}(W) = \mathcal{J}(\hat{W}P^{-\frac{1}{2}}) = \hat{\mathcal{J}}(\hat{W})$. The standard SGD updates the solution $\hat{W}$ for the projected loss with the equation below.

$$\hat{W}' = \hat{W} - \eta \nabla_{\hat{W}} \hat{\mathcal{J}}(\hat{W}), \tag{3}$$

where η is the learning rate and $\nabla_{\hat{W}} \hat{\mathcal{J}}(\cdot)$ is the gradient of loss $\hat{J}(\cdot)$ with respect to $\hat{W}$. Thus $\nabla_{\hat{W}} \hat{\mathcal{J}}(\cdot)$ has the same dimensions as $\hat{W}$.

In the neural network training, the weight matrix to learn is W. We expand Equation (3) with the expression of $\nabla_{\hat{W}} \hat{\mathcal{J}}(\hat{W})$ where $\nabla_{\hat{W}} \hat{\mathcal{J}}(\hat{W}) = \nabla_W \mathcal{J}(W)(P^{-\frac{1}{2}})^T$ and derive the following update formula for W. The derivation of the update formula is available in our supplementary material.

$$W' = W - \eta \nabla_W \mathcal{J}(W) P^{-1}, \tag{4}$$

where $\nabla_W \mathcal{J}(W) \in \mathbb{R}^{d_{out} \times d_{in}}$ is the gradient of loss $\mathcal{J}(\cdot)$ with respect to W. Updating the weight matrix W of the original problem using Equation (4) is equivalent to updating the solution $\hat{W}$ of the projected problem in the transformed space using Equation (3). In Kernel SGD, we use the kernel matrix H of the kernel machine as the projection matrix P. The positive semi-definite kernel matrix may transform the problem to a space where Kernel SGD is more likely to converge to a better solution. Hence Kernel SGD updates the weight W with the following formula which corresponds to the update of the projected weight in the space transformed by the kernel matrix.

$$W' = W - \eta \nabla_W \mathcal{J}(W) H^{-1}. \tag{5}$$

In the training of neural networks, a mini-batch of the training instances is commonly used. Therefore, we may not have all the training instances to construct the whole kernel matrix. In the mini-

batch setting, we use a subset of the training data, for example, a mini-batch of m training instances, to approximate the kernel matrix where H is then an $m \times m$ matrix.

Moreover, the matrix multiplication between the gradient matrix $\nabla_W \mathcal{J}(W)$ and the inverse kernel matrix H^{-1} constrains that the number of rows of kernel matrix should equal the number of columns of the gradient matrix $\nabla_W \mathcal{J}(W)$. When the above multiplication constraint is not satisfied, the inverse of kernel matrix is reshaped in the following ways: i) If $m < d_{in}$, we pad the inverse of kernel matrix with elements of zeros so that the number of rows of the kernel matrix satisfies the constraint; ii) If $m > d_{in}$, we remove the last $m - d_{in}$ rows and columns of the inverse of kernel matrix. In the supplementary material, we prove that the reshaping of the inverse kernel matrix guarantees the training converged.

3.2 Kernel SGD in Neural Network Training

We show the pseudo-code of Kernel SGD in Algorithm 1. In Kernel SGD, we first compute the kernel matrix H for each mini-batch (Line 4). The kernel matrix is stored as an $m \times m$ matrix in each mini-batch and the whole kernel matrix for all the mini-batches is thus an $n \times m$ matrix where n and m are the number of instances and the mini-batch size, respectively. Then we compute the inverse of the kernel matrix for each mini-batch with reshaping if necessary (Line 5). Next, in the backward propagation, Kernel SGD updates the weights of the network from the end to the beginning using Equation (5) with the corresponding inverse matrix. Moreover, the input of the last layer can be treated as the learned presentation of the input instance which corresponds to $\phi(\boldsymbol{x})$ in kernel machines. The last layer of the network can be regarded as the inference layer which has a similar decision function as general kernel machines. Thus, it is more natural to apply the update rule (5) only in the last fully connected layer, and the rest of the layers are updated using the original SGD (Line 7-11). We also conducted experiments to compare the performance of applying Kernel SGD in the last layer with applying it in all the intermediate layers. The results further confirm that Kernel SGD applied in the last layer is computation efficient and can perform more accurate prediction. Therefore we adopt Kernel SGD following this manner in our experiments.

Unlike the current second-order optimization and preconditioned SGD (Dauphin et al., 2015) which all need to update the Hessian matrix in each iteration, Kernel SGD uses fixed kernel matrix. Therefore, the computation cost for kernel matrix is rather small compared with the training of the whole network. We can further accelerate Kernel SGD by computing the kernel matrix in parallel with the training of neural networks. On the contrary, updating the Hessian matrix while training the networks is practically infeasible in most second-order optimization. Considering that the mini-batch size is much smaller than the total number of training instances, the storage cost of the kernel matrix of the mini-batches is also smaller which is linear in the number of instances, i.e., $\mathcal{O}(n \cdot m)$.

Algorithm 1: Neural network training with Kernel SGD

Input: Training instances X with labels $\boldsymbol{y}$, a neural network model M, and a kernel k.
Output: The optimized neural network model M^*

1 **foreach** *epoch* **do**
2 **foreach** *batch* **do**
3 $X_b \in \mathbb{R}^{m \times d}, \boldsymbol{y}_b \in \mathbb{R}^m \leftarrow$ sampleData(X,m) // Sample m instances for a batch
4 $H = [H_{ij}]_{m \times m} \leftarrow$ computeKernelMarix(X_b, k) // $H_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$
5 $H^{-1} \leftarrow$ computeInverseKernel(H) // H^{-1} is reshaped
6 $\mathcal{J} \leftarrow$ computeObjective($X_b, \boldsymbol{y}_b, M$)
7 **for** $t \leftarrow \mathcal{T}$ **to** 1 **do** // Back propagation
8 **if** $t == \mathcal{T}$ **then** // $\mathcal{T}$ is the total number of layers
9 $M' \leftarrow$update the weights in the layer t using Kernel SGD with Equation (5)
10 **else**
11 $M' \leftarrow$update the weights in the layer t using standard SGD
12 $M = M'$
13 **return** The optimized model $M^* \leftarrow M$

3.3 CONVERGENCE ANALYSIS

Here, we theoretically demonstrate the convergence guarantee of Kernel SGD optimization. In a general neural network for classification, the last layer is commonly a fully connected (FC) layer. We denote the weight matrix of the last layer by W which is an $n_c \times d_{in}$ matrix (i.e., $d_{out} = n_c$), where n_c is the number of classes. Without loss of generality, we consider standard SGD that feeds one instance into the network in each iteration. Suppose the instance $\boldsymbol{x}$ belongs to the i-th class. Then, the cross entropy loss can be defined as $\mathcal{J}(W) = -y_i \ln a_i$, where $a_i = e^{f_i} / \sum_{j=1}^{n_c} e^{f_j}$ and is the value of the activation function which is a softmax function; f_i is the output of the FC layer. The loss $\mathcal{J}(W)$ is a function of variable W and the weights of other layers can be treated as constants. We can rewrite the cross entropy loss with f_i as below.

$$\mathcal{J}(W) = -f_i + \ln \sum_{j=1}^{n_c} e^{f_j}. \tag{6}$$

The output of FC layer can be computed as $f_i = W_i G(\boldsymbol{x})$, where $G(\boldsymbol{x}) \in \mathbb{R}^{d_{in}}$ denotes the input to the FC layer and can be treated as the learned representation of $\boldsymbol{x}$; W_i is the i-th row in of the matrix W. In the equation of computing f_i, we omit the bias term, since it can be written as an element W_i.

Next, we give the convergence guarantee of Kernel SGD.

Convergence Theorem. *In the neural network with the last layer of a fully connected layer with softmax activation function, given the weight matrix W of the last layer and the corresponding updated weight matrix W' computed by Equation* (5)*, the cross entropy losses $\mathcal{J}(W')$ and $\mathcal{J}(W)$ satisfy the following inequality.*

$$\mathcal{J}(W') \leq \mathcal{J}(W). \tag{7}$$

Based on the Convergence Theorem, the loss decreases or stays unchanged as the training progresses using Kernel SGD. Hence we can conclude that the optimization using Kernel SGD is guaranteed to converge. Detailed proof of the Convergence Theorem is available in the supplementary material.

Next, we propose a Proposition to show that Kernel SGD is more likely to converge to a better solution.

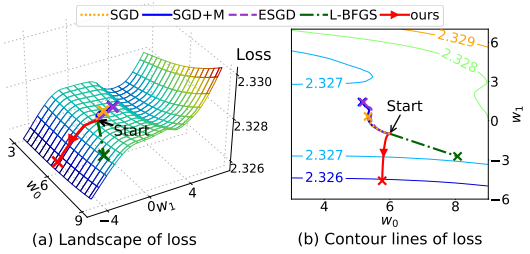
Proposition. *Let W^* and $\hat{W}^*$ be the minimums of the original and projected loss, respectively. Let W and $\hat{W}$ be the initial points in the original space and the projected space, respectively. Assume that the eigenvalues of kernel $H \in \mathbb{R}^{n \times n}$ are $\{\pi_1, \ldots, \pi_n\}$ and $\sup(\sum_{\mathrm{i}=1}^{\mathrm{n}} \pi_{\mathrm{i}}) = \frac{1}{\mathrm{n_c}}$. Then we have the inequality below.*

$$\left|\left|\mathrm{vec}(\hat{W}^* - \hat{W})\right|\right|_2 \leq \left|\left|\mathrm{vec}(W^* - W)\right|\right|_2, \tag{8}$$

where $\mathrm{vec}(W) = [W_{11} \ \ldots \ W_{n_c 1} \ \ldots \ W_{1 d_{in}} \ \ldots \ W_{n_c d_{in}}]^T$ *for a matrix W of dimension $n_c \times d_{in}$.*

The $\hat{W}$ in the transformed space can be treated to be equivalent to W in the original space. The Proposition indicates that the optimum is closer to the initial point in the transformed space. Hence Kernel SGD is more likely to find a better solution, thanks to the smaller gap between the optimum and the starting point in the transformed space. The proof of Proposition can be found in the supplementary material.

We can get some insight of the Proposition from observing the trajectories of the loss in Figure 1. We trained a shallow network which has three linear hidden layers and an FC output layer with the cross entropy loss. The *usps* data set was used to train the model. For ease of visualization, two randomly selected weights w_0, w_1 were updated in optimization while others were fixed. The loss decrease of trajectories using different optimization methods is illustrated in Figure 1. With the same initial weights, Kernel SGD found a better solution than the existing methods. The existing methods fell



(a) Landscape of loss (b) Contour lines of loss

Figure 1: Trajectories of different optimization methods. The cross markers indicate the end of the optimization.

into the saddle point or local minimum. More details and results can be found in Section 4 which further confirm our findings.

4 EXPERIMENTAL STUDIES

In this section, we study the key metrics (e.g., convergence and accuracy) of Kernel SGD and then summarize its overall performance. We also investigate the influence of batch size on Kernel SGD.

4.1 DATA SETS AND EXPERIMENTAL SETUP

We evaluate Kernel SGD on three types of data: tabular, image, and text. Table 1 shows the details of these data sets. The tabular data sets were downloaded from LIBSVM website[1]. The dimensions of *mnist* and *usps* are 780 and 256, respectively. We used *cifar10* and *SARS-CoV-2* (*S-CoV*) (Soares et al., 2020) as the image data. *S-CoV* contains RGB CT scans and is used to identify whether the patient is infected by the virus. Each image in the two data sets was first reshaped into 32×32 image and then normalized according to the mean and variance sampled, respectively. We used *IMDb* and *COVID-19-tweets* (*COV-tw*) (Nguyen et al., 2020) as the text data. *COV-tw* collects the English Tweets about COVID-19 and is labeled as informative or not. We took 20% of the training data to serve as the validation set except for *COV-tw*. In *COV-tw*, 1000 validation instances are provided. On tabular data, our method was evaluated using a simple two-linear-layer neural network which has 100 neurons in each hidden layer. For image classification, our method was tested on the ResNet-18 (He et al., 2016) network. To solve text classification problems, we used an LSTM network with pre-trained word vectors (Pennington et al., 2014).

Table 1: Data set information

type	data set	# training instances	# test instances	#classes
tabular	mnist	60,000	10,000	10
	usps	7,291	2,007	10
image	cifar10	50,000	10,000	10
	S-CoV	2,000	482	2
text	COV-tw	7,000	2,000	2
	IMDb	25,000	25,000	2

Baselines: Our method was compared with the widely used L-BFGS and Equilibrated SGD (ESGD) (Dauphin et al., 2015) optimization methods which both use the second-order information in training. ESGD adopts an equilibration preconditioner to reduce the condition number and escape saddle points. We used the implementation of ESGD published by Li (2018).

Experimental setup: The experiments were conducted on a machine with an Intel(R) Xeon(R) Silver 4210 CPU of 126 GB memory and two GeForce RTX 3090 GPUs running on a Linux OS. All the methods including the baselines and our method were implemented using PyTorch (Paszke et al., 2019). The size of history used in L-BFGS was set as 3 for limited memory. The mini-batch size was set to 64. L_2 regularization was used with its coefficient set to 10^{-4} following the setting in the studies (He et al., 2016; Huang et al., 2017). The same regularization coefficient for all the methods leads to a fair comparison. In Kernel SGD, to compute the kernel matrix, we used the radial basis function kernel (i.e., $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\gamma||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2)$). Through the whole experiment, the learning rate and the hyper-parameter γ for the kernel were selected from $\{0.1, 0.01, 0.001, 0.0001\}$. We terminated the training when the convergence conditions were satisfied, i.e., the change of loss is less than 1×10^{-4} in 3 consecutive epochs or the training reaches 500 epochs. We repeated each experiment five times to acquire average performance.

[1] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

4.2 KEY METRICS OF KERNEL SGD AND SANITY CHECK

Observation on loss decrease. To study the convergence of Kernel SGD, we first illustrate the trajectories of training loss in Figure 2. As shown in the figure, L-BFGS gets stuck at the saddle points or local minimum in the early stage. ESGD helps escape the saddle points but fails to converge to a better loss on the tabular and text data. Kernel SGD reaches the stable point in a small number of epochs while producing the lowest loss in all types of tasks. This indicates that Kernel SGD has the ability to escape the saddle points and can find a better solution in the non-convex problems.

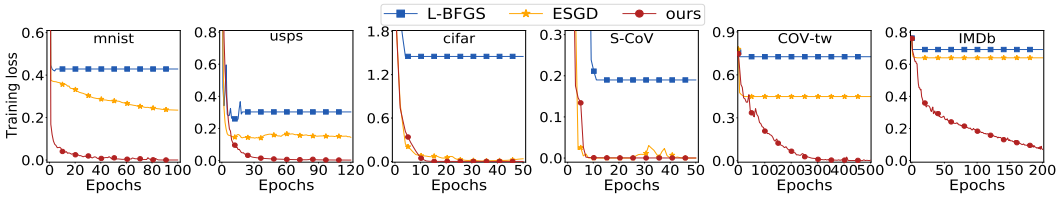


Figure 2: Training loss of each optimization method

Analysis on generalization. Here, we compare our method with second-order optimization to evaluate the generalization performance. The third and fourth columns of Table 2 list the training and test accuracy achieved by the best models of each optimizer. The best model was selected according to the highest validation accuracy. The values of selected learning rates are available in the supplementary material. Our method shows a remarkable generalization performance and can achieve the highest test accuracy on all the tasks. Especially on *IMDb*, Kernel SGD improves the test accuracy by around 28%. Moreover, Kernel SGD can mitigate overfitting and achieves a stable accuracy, which is demonstrated by the relatively small gaps between the test accuracy and training accuracy and the small variances. Note that we did not adopt pre-processing techniques such as random flipping on the tested data for fair comparison and thus the accuracy in Table 2 may be slightly different from those shown in other studies.

Analysis on convergence speed. We tested the convergence speed and recorded the convergence time in the fifth column of Table 2. Kernel SGD converges up to 30 times faster than second-order optimization baselines. For *Cov-tw* and *IMDb*, L-BFGS converges faster because it stops too early with a relatively large loss and underfits the problems as shown in Figure 2 and Table 2. Kernel SGD takes more epochs to converge to a better loss and thus needs more convergence time.

Analysis on memory cost. The memory for the Hessian matrix is represented in the sixth column of Table 2. We show the size of the whole kernel matrix in Kernel SGD which is an $n \times m$ matrix. In L-BFGS and ESGD, the Hessian matrix of neural networks are not explicitly computed. For L-BFGS, we recorded the memory consumption for storing the historical weights and gradients. In ESGD, we recorded the memory of the preconditioning matrix. Kernel SGD uses much smaller memory to store the kernel matrix in larger neural networks such as ResNet-18 and LSTM, for the computation is based on the training instances rather than the weights in the networks.

Sanity check. Although our main aim in this paper is to improve the second-order optimization methods, for a sanity check and for completeness, we further compare Kernel SGD with first-order optimizers which are mini-batch SGD and SGD with momentum (SGD+M). The parameter momentum was set as 0.9 for SGD+M. The results are shown on the last two columns of Table 2. Our method still outperforms the first-order optimizers in terms of generality on most tasks (i.e., 5 out of 6 tasks in total). Our method converges even faster than the first-order optimizers with image data.

4.3 OVERALL PERFORMANCE

We summarize the overall performance of different optimization methods in Figure 3. We compare these methods in terms of training accuracy, test accuracy, robustness, stability and convergence speed. The robustness indicates the gaps between the training and test accuracy. The stability is measured with the number of epochs which lead to the decrease of loss. Figure 3 shows that

Table 2: Comparison of the model accuracy (%), convergence time (sec.) and Hessian size (MB).

data set	Kernel SGD vs. second-order optimization					Kernel SGD vs. first-order optimization		
	optimizer	train. accuracy	test accuracy	converg. time	Hessian	optimizer	test accuracy	converg. time
mnist	ours	99.92 ± 0.17	**97.94±0.30**	**272±230**	11.72	ours	**97.94±0.30**	272 ± 230
	L-BFGS	87.70 ± 2.86	87.77 ± 2.82	555 ± 535	1.97	SGD	97.86 ± 0.07	**130±61**
	ESGD	93.90 ± 1.04	92.77 ± 0.52	8671 ± 9148	**0.18**	SGD+M	97.85 ± 0.21	2832 ± 2849
usps	ours	99.90 ± 0.09	**93.63±0.34**	**30±16**	1.42	ours	***93.63±0.34***	30 ± 16
	L-BFGS	93.74 ± 0.81	89.15 ± 1.12	888 ± 838	1.97	SGD	93.28 ± 0.20	**27±15**
	ESGD	96.04 ± 0.91	91.72 ± 0.70	37 ± 27	**0.18**	SGD+M	**94.17±0.29**	35 ± 28
cifar10	ours	100.00 ± 0.00	**83.30±0.22**	**475±32**	**9.77**	ours	**83.30±0.22**	**475±32**
	L-BFGS	59.53 ± 4.75	55.39 ± 4.13	3894 ± 467	468.92	SGD	82.59 ± 0.78	644 ± 73
	ESGD	99.79 ± 0.20	67.73 ± 0.38	9758 ± 587	42.63	SGD+M	83.23 ± 0.33	582 ± 19
S-CoV	ours	100.00 ± 0.00	**73.40±0.92**	**64±75**	**0.39**	ours	**73.40±0.92**	**64±75**
	L-BFGS	98.46 ± 0.75	70.00 ± 2.68	607 ± 376	468.92	SGD	68.13 ± 1.86	71 ± 13
	ESGD	99.99 ± 1.42	72.03 ± 2.72	199 ± 270	42.63	SGD+M	67.47 ± 3.24	74 ± 10
COV-tw	ours	90.21 ± 5.21	**77.55±2.07**	325 ± 156	**1.71**	ours	**77.55±2.07**	325 ± 156
	L-BFGS	51.84 ± 1.27	52.68 ± 0.43	**40±35**	125.36	SGD	74.17 ± 0.18	190 ± 148
	ESGD	79.68 ± 1.79	71.72 ± 0.78	2590 ± 51	11.40	SGD+M	77.03 ± 3.58	**151±98**
IMDb	ours	95.05 ± 2.75	**89.76±0.56**	5548 ± 1975	**4.88**	ours	**89.76±0.56**	5548 ± 1975
	L-BFGS	50.99 ± 1.30	50.89 ± 1.25	**610±725**	125.36	SGD	88.26 ± 1.73	**3404±2654**
	ESGD	63.68 ± 1.60	61.69 ± 1.89	7116 ± 656	11.40	SGD+M	83.01 ± 9.89	3877 ± 3269

Kernel SGD converges fast with high stability and robust generalization performance. Although SGD achieves good training and test accuracy, it performs less robust optimization. SGD+M and ESGD lacks robustness and stability. L-BFGS cannot achieve good accuracy.

4.4 Impact of Batch Size on Kernel SGD

As the kernel matrix used in Kernel SGD is correlated to the data in each batch, we varied the batch size and investigate the impact of batch size on Kernel SGD. We illustrate the test accuracy and convergence time in Figure 4. The decrease in batch size leads to a positive impact on the predictive accuracy in the data sets tested. This indicates that with less memory consumption for the kernel matrix, Kernel SGD can still achieve good predictive accuracy. When the batch size decreases, the convergence time increases for more vibrations in optimization with small batches. On the contrary, the convergence time for *S-Cov* increases with the increasing batch size because of its small number of training instances. For a larger batch size, each batch uses almost the whole data set to update the network only once. Therefore Kernel SGD needs more epochs and more time to converge in the training with *S-Cov* data set.

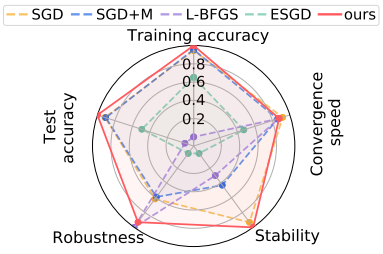


Figure 3: Overall performance.

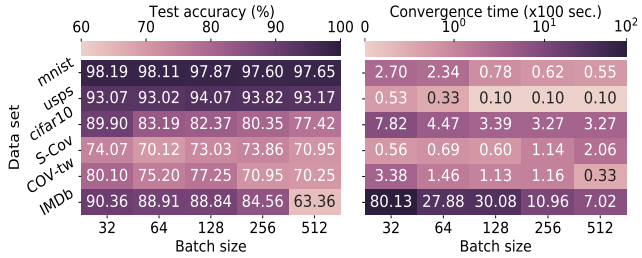


Figure 4: Impact of batch size on Kernel SGD.

4.5 Impact of Kernel Types on Kernel SGD

We evaluated Kernel SGD with other kernel functions which are linear kernel and polynomial kernel. The results show that Kernel SGD performs well with all the tested kernel functions. For example, Kernel SGD trained on *cifar10* achieved 83.76% 83.95% predictive accuracy which is similar to that achieved by Kernel SGD with RBF kernel. Moreover, we conducted experiments by replacing the

projection matrix with a random positive semi-definite (PSD) matrix. The training with a random PSD matrix shows a terrible performance on *cifar10* which has 10.07% predictive accuracy.

5 Related Work

We first present works that combine deep learning with kernel machines. Next we introduce the effort made to improve the second-order optimization in deep learning.

Deep Learning with Kernel Methods. Bo et al. (2010) used match kernels to build kernel descriptors which extract the patch-level features from the pixel features. Three kernel descriptors were designed to represent the gradient, color and shape features of images, respectively. Later, a convolutional multi-layer kernel was proposed (Mairal et al., 2014) which is a generalization of kernel descriptors and is approximated using a convolutional kernel network(CKN). Mairal (2016) improved the CKN with supervised learning and corresponding backpropagation procedures. Le et al. (2016) used deep learning to learn a kernel function for given data. Deep belief nets are used to maximize the similarity of the instances from the same class. The existing works either use the network to produce a new kernel or use the kernel function directly as a feature mapping to build the network. The second-order information in kernel methods is not well explored in the literature.

Second-Order Optimization. Due to the high computation cost, second-order optimization have been extensively studied in solving the over-parameterized deep learning problems. The quasi-Newton method including L-BFGS (Liu & Nocedal, 1989), Gauss-Newton (Schraudolph, 2002) and Kronecker-factored Approximate Curvature (K-FAC) (Martens & Grosse, 2015) is a class of Newton methods which computes an approximate Hessian matrix. Hessian-free methods (Martens, 2010) compute the Hessian-vector product in conjugate gradient without explicitly computing the Hessian matrix. ADAHESSAIN (Yao et al., 2021) integrates the first and second order momentum in optimization. Specifically, the second order momentum is updated with the Hessian diagonal in each iteration. Spatial averaging is applied to the Hessian diagonal to mitigate the noise of Hessian. Nonetheless, the recomputation of the derivatives is inevitable.

Preconditioned SGD. Preconditioned SGD transforms the gradient with a preconditioner to boost the optimization in ill-conditioned problems. Jacobi preconditioner is the one of the most popular preconditioners and is improved by LeCun et al. (2012) with Gauss-Newton matrix approximation. Li (2017) studied the properties of good preconditioners and proposed a new preconditioner estimation method. Staib et al. (2019) proposed to estimate the general preconditioners, for example, the preconditioner in RMSProp (Tieleman & Hinton, 2012) can be estimated by the inverse of the squared gradient. Gupta et al. (2018) designed the "Shampoo" algorithm which generates a separate preconditioner for each dimension of a tensor. Some preconditioning methods exploit the second-order information as the preconditioner (Schaul et al., 2013; Yao et al., 2018). Schaul et al. (2013) derived an adaptive learning rate schedule for SGD which uses the Hessian matrix to update the learning rate. Dauphin et al. (2015) proposed equilibrated SGD (ESGD) which applied an equilibration preconditioner. ESGD can compute the inverse of the absolute Hessian matrix efficiently while keeping the ability to escape the saddle points. Although approximation techniques are used, the frequent update of preconditioners remains the main obstacle to making preconditioning practical.

6 Conclusion

To improve the second-order optimization in the neural network training, we have proposed Kernel SGD which exploits the second-order information from kernel machines. Kernel SGD prominently reduces the computation and memory cost during the training of neural networks with second-order optimization. We provided a theoretical convergence guarantee for the training using Kernel SGD. Our experimental results on tabular, image and text data have shown that Kernel SGD achieves an overall superior performance than other existing optimization methods, especially in generalization and convergence speed. Our findings may encourage more research on this direction of incorporating kernel methods with deep learning.

REFERENCES

Mikhail Belkin, Siyuan Ma, and Soumik Mandal. To understand deep learning we need to understand kernel learning. *arXiv preprint arXiv:1802.01396*, 2018.

Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Kernel descriptors for visual recognition. In *Advances in neural information processing systems (NeurIPS)*, pp. 244–252, 2010.

Yann Dauphin, Harm De Vries, and Yoshua Bengio. Equilibrated adaptive learning rates for non-convex optimization. In *Advances in neural information processing systems (NeurIPS)*, pp. 1504–1512, 2015.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *International conference on machine learning (ICML)*, pp. 1842–1850. PMLR, 2018.

Yilong Hao, Kanishka Tyagi, Rohit Rawat, and Michael Manry. Second order design of multiclass kernel machines. In *2016 International joint conference on neural networks (IJCNN)*, pp. 3233–3240. IEEE, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 770–778, 2016.

Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, 36(3):1171–1220, 2008.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 4700–4708, 2017.

S Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural computation*, 15(7):1667–1689, 2003.

Linh Le, Jie Hao, Ying Xie, and Jennifer Priestley. Deep kernel: Learning kernel function from data using deep neural network. In *Proceedings of the 3rd IEEE/ACM International conference on big data computing, applications and technologies (BDCAT)*, pp. 1–7, 2016.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

Xi-Lin Li. Preconditioned stochastic gradient descent. *IEEE transactions on neural networks and learning systems (TNNLS)*, 29(5):1454–1466, 2017.

Xi-Lin Li. Preconditioner on matrix lie group for sgd. *arXiv preprint arXiv:1809.10232*, 2018.

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

Julien Mairal. End-to-end kernel learning with supervised convolutional kernel networks. In *Advances in neural information processing systems (NeurIPS)*, pp. 1399–1407, 2016.

Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid. Convolutional kernel networks. In *Advances in neural information processing systems (NeurIPS)*, pp. 2627–2635, 2014.

James Martens. Deep learning via hessian-free optimization. In *International conference on machine learning (ICML)*, volume 27, pp. 735–742, 2010.

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning (ICML)*, pp. 2408–2417, 2015.

Dat Quoc Nguyen, Thanh Vu, Afshin Rahimi, Mai Hoang Dao, Linh The Nguyen, and Long Doan. WNUT-2020 Task 2: Identification of Informative COVID-19 English Tweets. In *Proceedings of the 6th workshop on noisy user-generated text (W-NUT)*, pp. 314–318, 2020. URL https://www.aclweb.org/anthology/2020.wnut-1.41.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems (NeurIPS)*, pp. 8026–8037, 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. In *International conference on machine learning (ICML)*, pp. 343–351, 2013.

Bernhard Schölkopf, Ralf Herbrich, and Alex J Smola. A generalized representer theorem. In *International conference on computational learning theory*, pp. 416–426. Springer, 2001.

Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

Alex J Smola and Bernhard Schölkopf. *Learning with kernels*, volume 4. Citeseer, 1998.

Eduardo Soares, Plamen Angelov, Sarah Biaso, Michele Higa Froes, and Daniel Kanda Abe. Sars-cov-2 ct-scan dataset: A large dataset of real patients ct scans for sars-cov-2 identification. *medRxiv*, 2020. doi: 10.1101/2020.04.24.20078584. URL https://www.medrxiv.org/content/early/2020/05/14/2020.04.24.20078584.

Matthew Staib, Sashank Reddi, Satyen Kale, Sanjiv Kumar, and Suvrit Sra. Escaping saddle points with adaptive gradient methods. In *International conference on machine learning (ICML)*, pp. 5956–5965. PMLR, 2019.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Zeyi Wen, Zhishang Zhou, Hanfeng Liu, Bingsheng He, Xia Li, and Jian Chen. Enhancing svms with problem context aware pipeline. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 1821–1829, 2021.

Zhewei Yao, Amir Gholami, Daiyaan Arfeen, Richard Liaw, Joseph Gonzalez, Kurt Keutzer, and Michael Mahoney. Large batch size training of neural networks with adversarial training and second-order information. *arXiv preprint arXiv:1810.01021*, 2018.

Zhewei Yao, Amir Gholami, Sheng Shen, Kurt Keutzer, and Michael W Mahoney. Adahessian: An adaptive second order optimizer for machine learning. *AAAI (Accepted)*, 2021.